0-315-01203-X

National Library of Canada | Bibliothèque nationale du Canada

Canadian Theses Division | Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

49061

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

Zehra Meral Özsoyoğlu

Date of Birth — Date de naissance	Country of Birth — Lieu de naissance
July 6, 1951	Turkey

Permanent Address — Résidence fixe

Güniz Sokak, No: 14/14
Kavaklidere, Ankara, TURKEY

Title of Thesis — Titre de la thèse

Query Processing in Distributed Relational Databases

University — Université

Univ. of Alberta

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Ph. D.

Year this degree conferred — Année d'obtention de ce grade	Name of Supervisor — Nom du directeur de thèse
1980	Clement Yu

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date	Signature
August 21, 1980	Meral Özsoyoğlu

NL-91 (4/77)

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

THE UNIVERSITY OF ALBERTA

QUERY PROCESSING IN DISTRIBUTED RELATIONAL DATABASES

by

© ZEHRA MERAL OZSOYOGLU

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON ALBERTA CANADA

FALL 1980

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "Query Processing in Distributed Relational Databases", submitted by Zehra Meral Ozsoyoglu in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Supervisor

Supervisor in Residence

External Examiner

Date August 15, 1980

To my husband, Gültekin, and

my parents, Fehime and Kemal Yalçın

ABSTRACT

In processing distributed relational database queries, the cost of communication between sites is the dominant cost factor. It is generally agreed that the amount of data transferred determines this communication cost to a large extent. Thus it is desirable to minimize the amount of transmitted data.

Semi-join is a relational database operator which can be utilized to reduce the amount of data transmission in processing distributed queries. A class of queries, called tree queries, can always be answered using only semi-joins. This thesis is concerned with two related problems; determining tree query membership of a distributed query, and finding the optimum sequence of semi-joins to answer a tree query.

Distributed database queries involving join clauses with relational operators $\{>, \geq, =, \neq, \leq, <\}$ are considered. A canonical representation of a distributed query, called reduced join graph, is introduced. The query represented by a reduced join graph does not contain any redundant join clauses, and is equivalent to the original query. A conceptually simple and efficient algorithm is presented which determines whether or not such a query is a tree

query. For any tree query, the algorithm produces an equivalent query and its tree query graph so that the sequences of semi-joins to answer the original query can immediately be obtained. An implementation of the algorithm is outlined and its time and space requirements are established.

Distributed query optimization for tree queries is the second problem studied in this thesis. A class of tree queries with equi-join clauses are considered. In general, there are numerous semi-join sequences, called strategies, to evaluate a tree query, and these strategies may differ significantly in the amount of required data transmission. In order to find optimum strategy, first, a set of properties is given to eliminate the strategies that can never be the optimum; then a dynamic programming approach is used to find the optimum strategy among the potentially optimum ones.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Clement Yu, for his uninterrupted guidance and assistance during the course of this thesis, even though he was on leave in Chicago for the last two years. I have benefited from his insights, intuitions and encouragement as well as his criticisms, particularly on my writing. I am grateful to the members of my committee, Drs. Francis Chin (who has also been my supervisor-in-residence), John Tartar, Al Hussaini, Jeff Sampson and Bing Yao for taking time to read, criticize and comment on the thesis. I would also like to thank Dr. Len Schubert who has been my professor in two courses.

We have enjoyed the friendship of everybody in the Computing Science Department during our stay in Edmonton. I am grateful to Jennifer Penny and Jim Lamont for their moral support and careful proof reading of the thesis and an early draft of a portion of the thesis.The financial assistance from the Department of Computing Science and a Dissertation Fellowship from the University of Alberta are gratefully acknowledged.

Above all, special thanks must go to my husband and best friend, Tekin, for many many things. Without his moral support, patience, understanding and help, I would not have completed this thesis. Thank you Tekin.

TABLE OF CONTENTS

CHAPTER		PAGE
1:	INTRODUCTION	1
	1.1 Relational Databases	2
	1.2 Distributed Query Processing	8
	1.3 Overview and Outline of the Thesis	11
2:	SEMI-JOINS AND TREE QUERIES	15
	2.1 Definitions and Background	15
	2.2 Representation of Queries, Join Graphs	17
	2.3 Transitive Closure of Join Graph	22
	2.4 Equivalence of Queries	23
	2.5 Efficiency of Semi-joins	24
	2.6 Query Graphs, Tree and Cyclic Queries	25
3:	DETERMINING TREE QUERY MEMBERSHIP OF A DISTRIBUTED QUERY	29
	3.1 Introduction	29
	3.2 Reduced Join Graph	30
	3.3 A General Tree Query Membership Algorithm	38
	3.4 An Implementation of the Algorithm	53
	3.5 Summary	63
4:	DISTRIBUTED QUERY OPTIMIZATION FOR TREE QUERIES	65
	4.1 Introduction	65

4.2 Problem Formulation 72

4.3 Potentially Optimum Strategies 88

4.4 Optimization 110

4.4.1 Parameters Related to the Optimum Strategy 110

4.4.2 Optimum Strategy Fully Reducing a Specified Relation 111

4.4.3 Optimum Strategy Fully Reducing One of the Relations 126

4.4.4 An Example 129

5: CONCLUSIONS 133

BIBLIOGRAPHY 137

APPENDIX A: 140

A.1 Transitive Reduction of an Acyclic Join Graph . 140

A.1.1 Basic Definitions 140

A.1.2 Transitive Reduction 142

A.2 Computing the Transitive Reduction 147

LIST OF FIGURES

Figure		Page
1.1	An Example of a Relational Database	3
1.2	Relational algebraic Operations	5
2.1	A Query, its Query Graph and Join Graph	19
2.2	Tree and Cyclic Queries	28
3.1	Construction of Reduced Join Graph	33
3.2	Linked List Structure used by the Algorithm ...	56
4.1	Illustration for Lemma 4.4	78
4.2	Illustration for Lemma 4.5	78
4.3	A Strategy $T(R_r,S)$ and its Substrategies $T(R_i,S_i)$, i=1,2,...,t.	91
4.4	Illustration for Lemma 4.8	94
4.5	Illustration for Lemma 4.9	96
4.6	Illustration for Lemma 4.11	101
4.7	Illustration for Lemma 4.12	105
4.8	The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 1 ...	114
4.9	The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 2 ...	114
4.10	Substrategies of $t(R_r,S,a_{sr})$	114
4.11	The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 3 ...	117
4.12	The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 4 ...	117
4.13	Illustration for Lemma 4.14	121
4.14	Illustration of the m.n stages in computing the optimal Strategy over $S=S_{m,n,1}$	123

4.15 The Sizes and the Selectivities of the Relations 131

4.16 The Optimal Strategy $T(R_1,S)$ and the Strategy TG 131

A-1 The operators $\underline{U}$ and $\underline{\cap}$ 143

A-2 An Example for Algorithm T 150

CHAPTER 1

INTRODUCTION

Distributed database management systems (DBMSs) allow a collection of data to be stored at multiple locations and accessed as a single unified database. In applications requiring access to an integrated database from geographically dispersed locations, a distributed DBMS has the following major advantages over a conventional centralized DBMS [19]: (1) it is more reliable since it is supported by multiple computers at multiple locations, (2) it provides faster access by storing data at locations where it is frequently used, and (3) the capacity can be adjusted more easily to changing needs. These advantages and the growing field of applications of distributed databases have contributed to the need for the development of general purpose distributed DBMS. An overview of the technical problems associated with the development of a general purpose distributed DBMS and a survey of the research and development efforts to overcome these problems can be found in [19].

An important problem to be overcome in developing a general purpose distributed DBMS is to find efficient strategies for query processing. Since a distributed DBMS allows its users to access geographically dispersed data as a single unified database, user queries may reference data located at multiple sites of the distributed database. Processing such a query requires the transmission of the data between different sites via the communication network. The data transmission and the communications between sites are the primary sources of delay in query processing in a distributed database environment. In contrast, the primary sources of delay in a centralized database environment are secondary accesses and CPU time. Hence query processing strategies for centralized DBMS work poorly in processing distributed queries. This thesis is concerned with the problem of designing efficient strategies for processing distributed database queries.

1.1 Relational Databases

In this thesis, a relational database [6,8,9] with relations distributed over different sites is assumed. Below, the terminology associated with the relational database model is briefly reviewed.

A relational database is a collection of two dimensional tables called relations (e.g. Fig.1.1). The columns of each table are labeled by a set of distinct

SUPP:

S#	Sname	City
S_1	Bill	Chicago
S_2	Joe	Edm.
S_3	John	Edm.
S_4	Doe	Chicago

ORD:

J#	S#	P#	Qty
J_1	S_2	P_1	500
J_2	S_2	P_2	200
J_2	S_1	P_1	150
J_3	S_1	P_1	200

PROJ:

J#	City
J_1	Chicago
J_2	Edmonton
J_3	Ankara

STOCK:

S#	P#	Qoh
S_1	P_1	400
S_2	P_2	500
S_2	P_3	100

Figure 1.1. An Example of a Relational database

attributes. The values of the entries in a column of a relation are drawn from a set of values called the domain of the attribute associated with that column. Each row of a relation is called a tuple. Hence a relation can also be viewed as a set of tuples, each tuple having the same set of attributes. A relation R with a set of attributes $\{a_1,\ldots,a_n\}$ is denoted by $R(a_1,\ldots,a_n)$. An attribute a of a relation R is denoted R.a. The cardinality of a relation R, denoted $|R|$, is the number of tuples in R.

The relational query language used in this thesis is relational algebra [7]. The relational algebra is a collection of operations that deal with relations yielding new relations as a result. The relational algebra operations used in this thesis are projection, selection, join and semi-join.

Projection: Given a relation R on a set of attributes X, the projection of R over a set of attributes $T=\{t_1,\ldots,t_k\}$, $T\subseteq X$, selects the columns of R labelled by the elements of T and then eliminates the duplicate rows. It is denoted by R[T] or $R[t_1,\ldots,t_k]$ (e.g. Fig. 1.2(a)).

Selection (also called theta-selection or restriction): The selection of a relation R selects rows from R that satisfy a given qualification q, and is denoted by $z_q(R)$ (e.g. Fig. 1.2(b)). The qualification q is a conjunction of selection clauses of the form (R.a θ constant) where a is an attribute

SUPP[City]:

City
Chicago
Edm.

(a) Projection

z_q(STOCK):

S#	P#	Qoh
S_1	P_1	400
S_2	P_3	100

q=(STOCK.Qoh < 500)

(b) Selection

STOCK $\underline{\text{Join}}_q$ ORD:

J#	S#	P#	Qty	Qoh
J_2	S_2	P_2	200	500
J_2	S_1	P_1	150	400
J_3	S_1	P_1	200	400

q = (ORD.S# = STOCK.S#).and.(ORD.P# = STOCK.P#)

.and.(ORD.Qty ≤ STOCK.Qoh)

STOCK $\underline{\text{semi-join}}_q$ ORD:

S#	P#	Qoh
S_2	P_2	500
S_1	P_1	400

(c) Join and Semi-join

of R_i, $\theta \in \{>, \geq, =, \neq, <, \leq\}$ and is applicable to the constant and the domain of a.

Join: The join of two relations R_i and R_j, denoted R_i join$_q$ R_j, is obtained by concatenating the tuples of R_i and the tuples of R_j for which the join qualification q is true, (e.g. Fig. 1.2(c)). The join qualification q is a conjunction of join clauses of the form $(R_i.a \ \theta \ R_j.b)$ where a and b are attributes of R_i and R_j respectively and are defined on a common domain. The symbol θ in the join clause is one of the relational operators $\{>, \geq, =, \neq, <, \leq\}$ and is applicable to the common domain of the attributes a and b. The join of R_i and R_j on qualification q is called equi-join if all the relational operators in q are equality, i.e. q is a conjunction of equi-join clauses. since equi-join yields a result which necessarily contains two sets of identical columns, the redundant columns are eliminated after the equi-join. Similarly, join of R_i and R_j on q contains two identical columns for each equi-join clause in q. The notation R_i join$_q$ R_j denotes the result of the join after the redundant columns corresponding to equi-join clauses in q are eliminated.

Semi-join: The semi-join of a relation R_i by a relation R_j on a join qualification q, denoted R_i semi-join$_q$ R_j, or $R_i <\!\!-\!\!- R_j$, is the projection of R_i join$_q$ R_j over the attributes of R_i, i.e. R_i semi-join$_q$ R_j = (R_i join$_q$ R_j)[attributes of R_i].

A query is a function applied to relations to retrieve information from the database [24]. In this thesis, we consider queries which consist of a qualification and a target list.

Example 1.1: Consider the database given in Fig. 1.1 and the following query statement:

> "List the names of suppliers who have orders from the projects that are located in the same city as the suppliers and the quantity ordered is greater than 500."

The qualification of this query can be expressed by the following conjunction of selection and join clauses:

q=(SUPP.S#=ORD.S#) .and. (ORD.J#=PROJ.J#)
.and. (SUPP.City=PROJ.City) .and. (ORD.Qty > '500')

The target list of the query consists of an attribute, namely, SUPP.Name. #

Using high level query languages, users formulate queries in terms of the content of the information to be retrieved without reference to system oriented complexities. Since such queries describe which information is to be retrieved rather than how it is to be retrieved, it is DBMS's task to find efficient query processing strategies.

A number of algorithms have been proposed to optimize

the query processing or to achieve a satisfactory degree of efficiency in processing relational queries in a centralized database [18, 21, 27, 28]. The algorithms in [21, 27, 28] are based on heuristics (e.g. elimination of common subexpressions, etc.) that transform a query into an equivalent one which is easier to evaluate but not necessarily the optimal one over all possible evaluations of the query. In [28], Yao analyses the alternative ways to evaluate queries involving selection join and projection.

Optimization of centralized relational database queries involves two major difficulties. First, there is an exponential number of different formulations of a given query among which the one that can be evaluated in minimum time is to be chosen. Second, it is difficult to define the cost function in terms of complex properties of the storage structures and the access mechanisms in a centralized database system. A general treatment of query optimization in a relational database can be found in chapter 6 of [24].

Though the research on optimizing relational queries in a centralized database is an ongoing one, the query processing strategies for centralized DBMS do not extend well to distributed DBMS due to the differences in cost criterion, as discussed in the following section.

1.2 Distributed Query Processing

As an example of a distributed database query, consider the query given in example 1.1, and assume that the relations referenced by the query reside at different sites. Processing such a query involves transmission of relations between sites as well as operations such as join of the two relations residing at the same site, projection and selection which do not require any data transmission. Query processing which do not require any data transmission is called local processing, and all the inter-site data transmissions are called communications. In distributed query processing, it is common to assume that the cost of communications dominates the cost of local processing [3, 11, 13, 19, 26]. Hence, minimizing the amount of data to be transmitted in processing a distributed query is of prime importance [19]. That is, if each relation is stored entirely at one site then the costs of selection and projection operations are negligible since they are unary operations. If the two relations to be joined reside at different sites then such a join is the most costly operation since one of the relations is to be transmitted to the site of the other relation. On the other hand, a semi-join involving two relations at different sites can be computed by transmitting only the projection of one of the relations over its common joining attributes to the site of the other relation. Since a semi-join (e.g. R_i semi-join$_q$ R_j

can be computed with much less data transmission than the corresponding join (e.g. $R_i \underline{\text{join}}_q R_j$ which is computed by transmitted R_j to the site of R_i), semi-joins can be effectively utilized in distributed query processing to reduce the amount of data transmission. In [13, 26] semi-joins are used to reduce the cost of subsequent join operations between relations. In this thesis, the semi-joins are further investigated for processing distributed database queries efficiently. Below is a brief review of the previous research on distributed database query processing.

A considerable amount of research has been done in minimizing or reducing the amount of data transmission in distributed query processing [3, 4, 11, 13, 26]. For query processing in SDD-1 system [20], Wong has proposed an algorithm [26] which obtains a local optimum solution. Wong's algorithm can be summarized in three steps:

(1) Perform as much local processing of the query as possible before any data transmission,

(2) Find the initial feasible strategy, i.e., the strategy with minimum cost among all strategies which move all the relations referenced by the query to one of the sites in parallel without any intervening local processing.

(3) Refine the initial feasible strategy successively by local hill climbing techniques.

An improved version of Wong's algorithm using branch and bound techniques is given in [11].

Hevner and Yao [13] studied the problem under the independence assumption of domains in a relational distributed database. Two cost criteria are used: "total time" and "response time" corresponding to cost of data transmissions with and without considering the parallel transmissions respectively. Their approach results in an optimum strategy (for both cost criteria) for simple queries[+] involving single attribute relations. For more general queries, their algorithm [13] serves as a heuristic.

More recently, Bernstein and Chiu [3] classified queries into two disjoint classes: tree queries and cyclic queries. They showed that tree queries can be evaluated with a small number of semi-joins whilst evaluating cyclic queries involves more elaborate data transfer.

Since tree queries can be answered by semi-joins, it is important to be able to determine whether a given query is a tree query or not. An algorithm for this task was presented in [3]. It has been generalized to allow more than one common joining attribute between any two relations [4, 29]. The queries considered in [3, 4, 29] involve equi-join clauses only.

[+] A simple query is a conjunction of equi-join clauses such that after the local processing each relation has a single attribute, and that attribute is common to all the relations referenced by the query.

1.3 Overview and Outline of the Thesis

In this thesis, the problem of determining whether a given distributed query is a tree query or not is studied by considering more general queries. An efficient tree query membership algorithm is presented for queries whose qualifications are conjunctions of join and selection clauses with relational operators $\{>, \geq, =, \neq, <, \leq\}$. Moreover, there is no restriction on the number of attributes that a relation involved in the query may have.

In general, there are numerous semi-join sequences that can be used to evaluate a given tree query, and these semi-join sequences may differ significantly in the amount of required data transmission. Finding the optimum sequence of semi-joins to evaluate tree queries is an important special case of the general distributed query optimization problem for which there is no known optimum solution. Distributed query optimization for tree queries is the second problem studied in this thesis. An efficient dynamic programming algorithm that finds the optimum sequence of semi-joins for a class of tree queries is presented. For some tree queries the optimum sequence of semi-joins is also the optimum procedure to answer the query. As an example consider a tree query such that fully reducing a relation R_r referenced by the query is sufficient to answer the query and R_r resides at the site where the answer of the query is

required. Clearly, the optimum sequence of semi-joins which fully reduces R_r is also the optimum way to answer such a query. However, the optimum sequence of semi-joins may not be optimum among all possible procedures (utilizing joins as well as semi-joins) to answer a distributed tree query in general. Nevertheless, the optimum sequence of semi-joins for a tree query is likely to be an efficient procedure to answer the query if not optimum.

Chapter 2 gives the motivations and the definitions that will be utilized in the remainder of the thesis. The distinction between tree and cyclic queries and the importance of semi-joins are discussed.

In Chapter 3, a canonical representation of a distributed relational database query, called reduced join graph, is introduced. The query represented by a reduced join graph does not contain any redundant join clauses, and is equivalent to the original query. The main result of Chapter 3 is a conceptually simple and efficient algorithm for determining the tree query membership of a distributed query, which may involve join and selection clauses with relational operators $\{>, \geq, =, \neq, <, \leq\}$. For tree queries, the presented algorithm produces an equivalent query for which a sequence of semi-joins to evaluate the query can immediately be constructed. An efficient implementation of the algorithm is also given together with its time and space complexities.

Chapter 4 studies the distributed query optimization for a class of tree queries. The tree queries considered in this chapter are conjunctions of equi-join clauses such that no two relations referenced by the query have more than one join attribute in common. For such a tree query, there is an exponential number of semi-join sequences, each of which can be used to evaluate the query. A set of properties are presented that must be satisfied by a potentially optimum strategy. These properties help to discard the strategies that can never be the optimum, so that the number of strategies to be considered in finding the optimum is reduced considerably. A recursive algorithm is then presented to find the optimum sequence of semi-joins. Some practical considerations of size estimations and the related difficulties are also discussed. Usually strong assumptions are required to estimate the sizes of intermediate relations which are produced while evaluating a query. However the optimization results of Chapter 4 are not based on the method used for size estimation and are applicable to any reasonable size estimation method.

Finally Chapter 5 summarizes the significance of the results obtained in this thesis and discusses possible avenues of further research.

CHAPTER 2

SEMI-JOINS AND TREE QUERIES

2.1 Definition and Background

In a distributed database, data is usually dispersed over the sites of the database in a redundant manner so as to improve the reliability and the responsiveness of the overall system [19]. However the problem of maintaining and managing redundant copies of the data is considered to be a different problem than that of retrieving distributed data from the database [26] and is not dealt with in this thesis. Hence, throughout this thesis, a relational distributed nonredundant database is assumed. Furthermore, each relation is assumed to reside in a single site, i.e. the relations are not fragmented among several sites, and the network is assumed to be fully connected. These two assumptions are also common to [3, 4, 11, 13, 26].

The local processing costs are assumed to be negligible in comparison to communications cost, and the communications cost is assumed to be determined by the amount of data

transmission. The aim is to minimize the amount of data to be transmitted between sites in processing a distributed query.

A query Q is assumed to be a conjunction of join clauses of the form (R_i.a θ R_j.b) where a and b are the attributes of the relations R_i and R_j respectively and $\theta \in \{>, \geq, =, \neq, <, \leq\}$ unless it is stated otherwise. The selection clauses (e.g. (R_i.a θ constant)) are omitted since such clauses can be handled by local processing. The exclusion of disjunctions is also justifiable (for the tree query membership problem) since disjunctions can be handled by transforming the qualification of the query into disjunctive normal form, and treating each conjunction as the qualification of a separate query. The target lists are omitted for notational convenience, however, the analysis will later be extended to allow queries with target lists. Other query constructs such as quantifiers, tuple variables and aggregate functions will not be addressed in this thesis. For notational convenience, each relation referenced by the query is assumed to reside at a different site. Since the objective is to minimize the amount of intersite data transmission, the analysis in this thesis is restricted to the class of queries which is not subject to local processing.

2.2 Representation of Queries, Join Graphs

A query Q is represented by a join graph, JG_Q, which is a directed graph whose vertices are the join attributes involved in Q and whose arcs are the join clauses of Q. Since $a \leq b$ and $a < b$ are the same as $b \geq a$ and $b > a$ respectively, it is sufficient to consider the join clauses of the form $(R_i.a\ \theta\ R_j.b)$ in Q, where $\theta \in \{>, \geq, =, \neq\}$. A join clause $(R_i.a\ \theta\ R_j.b)$, $\theta \in \{>, \geq, =\}$, in Q is represented by an arc from $R_i.a$ to $R_j.b$ in JG_Q (the operator "$\neq$" forms a special case). To distinguish between join clauses with different relational operators, three different types of arcs are used in JG_Q: type-1 arcs representing "$\geq$" join clauses, type-2 arcs representing ">" join clauses, and type-3 arcs representing "$\neq$" join clauses. A type-i arc, $1 \leq i \leq 3$, in JG_Q from $R_i.a$ to $R_j.b$ is denoted by $(R_i.a, R_j.b)_i$. Note that unlike type-1 or type-2 arcs a type-3 arc does not have direction, i.e., $(R_i.a, R_j.b)_3$ is the same as $(R_j.b, R_i.a)_3$, and either one of them can be used to represent the join clause $(R_i.a \neq R_j.b)$. Since $(a=b)$ is the same as $(a \geq b)$ and $(b \geq a)$, an equi-join clause $(R_i.a = R_j.b)$ in Q is represented by the two type-1 arcs, $(R_i.a, R_j.b)_1$ and $(R_j.b, R_i.a)_1$, in JG_Q. More precisely, the join graph $JG_Q(V,A)$ of a query Q is a labelled directed graph where

$V = \{R_i.a \mid a \text{ is a join attribute of } R_i \text{ in } Q\}$

$A = A_{type\text{-}1} \cup A_{type\text{-}2} \cup A_{type\text{-}3}$

$A_{type\text{-}1} = \{(R_i.a, R_j.b)_1 \mid (R_i.a \geq R_j.b) \text{ is in } Q\} \cup$
$\{(R_i.a, R_j.b)_1, (R_j.b, R_i.a)_1 \mid (R_i.a = R_j.b)$
$\text{is in } Q\}$

$A_{type\text{-}2} = \{(R_i.a, R_j.b)_2 \mid (R_i.a > R_j.b) \text{ is in } Q\}$

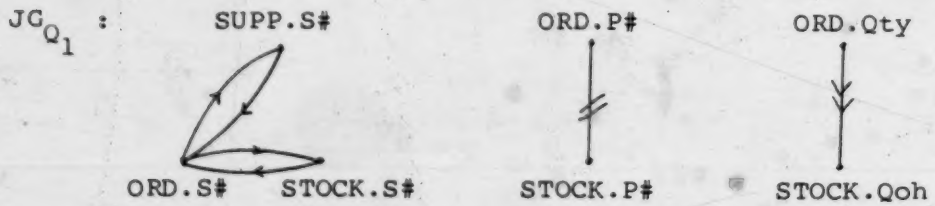
$A_{type\text{-}3} = \{(R_i.a, R_j.b)_3 \mid (R_i.a \neq R_j.b) \text{ is in } Q\}.$

Schematically, a type-1 arc, $(u,v)_1$, is shown as u-->-v, a type-2 arc, $(u,v)_2$, is shown as u-↠-v, and a type-3 arc, $(u,v)_3$, is shown as u-↛-v. An example of a query and the corresponding join graph is given in Fig. 2.1. For every join graph there is a unique query corresponding to it if the ordering of clauses in the query is not taken into consideration. Thus there is a one-to-one correspondence between queries and join graphs.

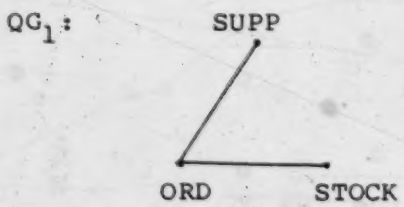
A join graph JG_Q corresponding to a query Q may contain a number of connected components, each being a directed graph with at least two vertices and containing type-1, type-2 and/or type-3 arcs. It is possible that there may be two or more different types of arcs from u to v in JG_Q, where u and v are any two distinct vertices. In such cases, all of the arcs from u to v can be replaced by a single arc, namely $(u,v)_2$, as follows. Suppose both arcs $(u,v)_1$ and $(u,v)_2$ are in JG_Q. This occurs when Q contains both clauses $(u \geq v)$ and $(u > v)$. Since Q is a conjunction of clauses, $(u \geq v)$ .and. $(u > v)$ is equivalent to $(u > v)$. Thus, if JG_Q contains

Q_1: (SUPP.S# = ORD.S#).and.(ORD.S# = STOCK.S#
.and.(ORD.P# ≠ STOCK.P#).and.(ORD.Qty > STOCK.Qoh)

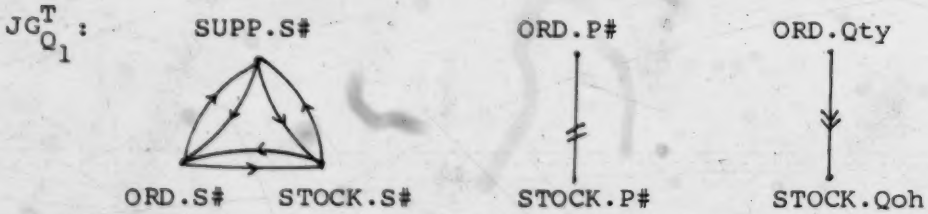
(a) A Query Q_1



(b) Join Graph of Q_1



(c) Query Graph of Q_1



(d) Transitive closure of JG_{Q_1}

Figure 2.1. A Query, its Query Graph and Join Graph.

both a type-1 arc(u,v) and a type-2 arc (u,v), the type-1 arc (u,v) is redundant and can be eliminated. Similarly, if JG_Q contains both arcs $(u,v)_2$ and $(u,v)_3$ then $(u,v)_3$ is redundant and can be eliminated. On the other hand, if JG_Q contains both arcs $(u,v)_1$ and $(u,v)_3$ then $(u,v)_1$ and $(u,v)_3$ can be replaced by the arc $(u,v)_2$ since $(u\geq v)$.and.$(u\neq v)$ is equivalent to $(u>v)$. In general, a type-2 arc $(u,v)_2$ is said to dominate type-1 or type-3 arcs from u to v, and if there are two or more different arcs in JG_Q from u to v then all of the arcs from u to v are replaced by the dominating arc, $(u,v)_2$. Thus, with no loss of generality, for any two distinct vertices, u,v, in a join graph, JG_Q, there is at most one arc from u to v.

A path in JG_Q from vertex u to vertex v is a sequence of distinct arcs $a_1,\ldots,a_p$, $p \geq 1$, such that there exists a corresponding sequence of vertices $u=v_0,v_1,\ldots,v_p=v$ satisfying $a_{k+1}=(v_k, v_{k+1})_t$ is in JG_Q for $0 \leq k \leq p-1$, where $1\leq t\leq 3$ for $p=1$, and $1\leq t\leq 2$ for $p\geq 2$. The motivation behind the definition of a path in JG_Q is to establish the join clauses which may or may not be in Q but are implied by Q. Suppose JG_Q contains the arcs $(u,w)_1$ and $(w,v)_3$, where u,v,w are three distinct vertices in JG_Q. The conjunction of the join clauses corresponding to these arcs, i.e., $(u\geq w)$ .and.$(w\neq v)$, does not imply any relation between u and v. Thus, the arcs $(u,w)_1$ and $(w,v)_3$ do not form a path from u to v. In general, no type-3 arc can be in a path of length

≥ 2 in JG_Q. A path is called a type-1 path if all the arcs in the path are type-1 arcs. If there are one or more type-2 arcs in a path of length ≥ 2 then the path is called a type-2 path. The type of a path of length 1 in JG_Q is the type of the only arc in the path. Thus, a path of length ≥ 2 is either type-1 or type-2.

A cycle in JG_Q is a path of length at least 2, which begins and ends with the same vertex. Observe that a self loop, i.e. an arc of the form (u,u) in JG_Q, is not a cycle. Since any join clause in Q corresponding to a self loop in JG_Q is redundant, we consider queries which do not have self loops in their join graphs. A join graph is called acyclic iff it contains no cycles.

Since Q is a conjunction of join clauses, and $\geq$ is transitive, a type-1 path in JG_Q from u to v implies that the join clause $(u \geq v)$ must be satisfied. Similarly, by transitivity, a type-2 path in JG_Q from u to v implies that the join clause $(u > v)$ must be satisfied. It is clear that no join clause of the form $(u \neq v)$ can be implied in this manner. However, a type-1 path of length ≥ 2 from u to v and the arc $(u,v)_3$ implies that the join clause $(u > v)$ is to be satisfied (by transitivity and dominance). For notational convenience, we call a type-1 path, $u=w_0,w_1,\ldots,w_n=v$, with the arcs $(w_{i-1},w_i)_1$, $1 \leq i \leq n$, $n \geq 2$, in JG_Q, an implied type-2 path from u to v if $(w_s,w_t)_3$ is in JG_Q, where $1 \leq s,t \leq n$, $2 \leq |s-t| \leq n$. Now consider a cycle in JG_Q incident with two vertices u and v.

one path from u to v implies $u \geq v$ and the other from v to u implies $v \geq u$. If one or more arcs in the cycle are type-2 arcs then at least one of the inequalities is a strict one; a contradiction. Thus, it is sufficient to consider those queries which have no type-2 arcs in cycles. Since $u \geq v$ and $v \geq u$ implies $u=v$, a cycle in JG_Q represents a conjunction of join clauses of the form $(u=v)$ between every pair of vertices, u,v, incident with the cycle. Clearly, a type-3 arc in JG_Q between any two vertices that are incident with the same cycle is a contradiction, i.e., $(u \neq v)$.and.$(u=v)$. Thus, it is sufficient to consider those queries that have only type-1 arcs within cycles in their join graphs.

2.3 Transitive Closure of Join graph

Let $JG_Q(V,A)$ be the join graph of a query Q where V is the set of vertices and A is the set of arcs. Informally, the transitive closure, JG_Q^T, of JG_Q is a join graph that represents all the join clauses that are in Q or implied by Q. Let u and v be two vertices in JG_Q. A type-1 arc $(u,v)_1$ is said to be implied by Q if there is a type-1 path in JG_Q from u to v such that the path does not use any arc from u to v. Similarly, a type-2 arc $(u,v)_2$ is implied by Q if there is a type-2 (or implied type-2) path in JG_Q from u to v such that the path does not use any arc from u to v. Moreover, a type-3 arc $(u,v)_3$ is said to be implied by Q if JG_Q contains an arc $(u,w)_3$, for some other vertex w, and a cycle incident with w and v.

Formally, the transitive closure of JG_Q is $JG_Q^T(V,A^T)$ where A^T is obtained by successively adding all implied arcs to A and replacing multiple arcs between two vertices by dominating arcs (see Fig. 2.1 and Fig. 3.1). Observe that for any arc $a=(u,v)_i$, $1 \leq i \leq 3$, in JG_Q, JG_Q^T contains exactly one arc from u to v, which either is the same as a or dominates a. JG_Q^T does not contain any self loops.

2.4 Equivalence of Queries

Two queries, Q_1 and Q_2, are equivalent if their answers[+] are the same irrespective of the contents of the relations [3]. Obviously, two queries having the same join graph are equivalent. The following lemma gives the necessary and the sufficient condition for the equivalence of queries in terms of join graphs.

Lemma 2.1: Q_1 is equivalent to Q_2 iff $JG_{Q_1}^T = JG_{Q_2}^T$.

Proof: Let $JG_{Q_1}^T = JG_{Q_2}^T$, and Q_1' denote the query represented by $JG_{Q_1}^T$. From the definition of transitive closure, any clause which is in Q_1' but not in Q_1 is implied by Q_1. But Q_1' contains all the join clauses in Q_1 except those which are dominated. Thus Q_1 is equivalent to Q_1'. Similarly, Q_1' is equivalent to Q_2 since $JG_{Q_1}^T$

[+] In general, the answer of a query is the set of tuples described by the qualification and the target list of the query. Since queries under consideration do not have target lists, the answer of a query in this context refers to the set of tuples (of all the relations referenced by the query) satisfying the qualification.

$= JG^T_{Q_2}$. Therefore Q_1 is equivalent to Q_2.

Suppose $JG^T_{Q_1} \neq JG^T_{Q_2}$. Since each connected component in a join graph contains at least two vertices and is connected, $JG^T_{Q_1}$ and $JG^T_{Q_2}$ must differ on at least one arc. This implies the existence of a join clause satisfied by the qualification of one but not the other query. Therefore, the answers of the two queries can not be the same, irrespective of the contents of the relations, so Q_1 is not equivalent to Q_2. #

2.5 Efficiency of Semi-joins

For an equi-join query Q, R_i semi-join$_q$ R_j can be computed by transferring only the projection of R_j over its attributes that appear in q [3], where q is the join qualification over R_i and R_j. If q contains '$\neq$' join clauses as well as equi-join clauses then it is evident that the data transfer required to compute R_i semi-join$_q$ R_j is also the projection of R_j over its attributes in q.

Suppose there is only one join clause between R_i and R_j in Q. Let this clause be $(R_i.a > R_j.b)$ or $(R_i.a \geq R_j.b)$. Then it is easy to verify that the data transfer required to compute R_i semi-join$_q$ R_j is only the minimum value in the column b of R_j. Similarly, in order to compute R_j semi-join$_q$ R_i, it is sufficient to transfer only the maximum value in column a of R_i. However, if there are several attributes of R_j appearing in the join qualification

q then, in order to compute R_i semi-join$_q$ R_j, it may be necessary to transfer the projection of R_j over its attributes in q to the site of R_i. Thus, for queries involving join clauses with relational operators {$\geq$, $>$, $=$, $\neq$}, the data transmission required to perform R_i semi-join$_q$ R_j is restricted to the projection of R_j over its attributes in q.

Clearly, a semi-join (unlike join) always reduces the size of the relation on which it is performed. That is, the number of tuples in the relation resulting after R_i semi-join$_q$ R_j is equal to or less than the number of tuples in R_i. Consequently, semi-joins can be effectively used to reduce the amount of data transmission in processing a distributed query. Furthermore tree queries, to be defined next, can be answered using only semi-joins [3].

2.6 Query Graphs, Tree and Cyclic Queries

Given the join graph JG_Q of a query Q, we want to examine whether it is possible to answer Q by using semi-joins. This is facilitated by defining a query graph. The query graph QG of a given query Q is an undirected graph whose vertices are the relation names referenced in Q, and whose edges indicate the data transfer between the relations. More formally, the query graph QG(V,E) [29] of a query Q is an undirected graph where

V = set of all relation names referenced in Q,

E = $\{(R_i, R_j) \mid i \neq j$ and there is an arc in JG_Q between some attribute of R_i and some attribute of $R_j\}$.

By definition no self loop (i.e., an arc of the form (R_i, R_i)) exists in a query graph. An example of a query graph is given in Fig. 2.1.

For the remainder of the thesis, it is assumed that every query has a connected query graph. This is not a restrictive assumption since otherwise the query is a conjunction of subqueries (whose query graphs correspond to the connected components in the query graph) each of which can be processed independently [3]. Furthermore, if the query graph of Q is connected, then any query equivalent to Q also has a connected query graph, as is shown by the following lemma.

Lemma 2.2: Let Q be a query whose query graph, QG, is connected. Then the query graph of any query that is equivalent to Q is also connected.

Proof: Consider any edge (R_i, R_j) in QG. The join graph JG_Q of Q must contain at least one arc incident with some attribute of R_i and some attribute of R_j. Let this arc be $(R_i.a, R_j.b)_k$, $1 \leq k \leq 3$. Let Q_1 be a query equivalent to Q, and JG_{Q_1} and QG_1 denote its join graph and query graph respectively. From Lemma 2.1, $JG_Q^T = JG_{Q_1}^T$, so JG_{Q_1} contains at least one path whose end vertices are $R_i.a$ and $R_j.b$. Hence, QG_1 contains a path from R_i to R_j. This shows

that if any two vertices are adjacent in QG then they are connected in QG_1. Since both QG and QG_1 have the same set of vertices and QG is connected, this implies QG_1 is also connected. #

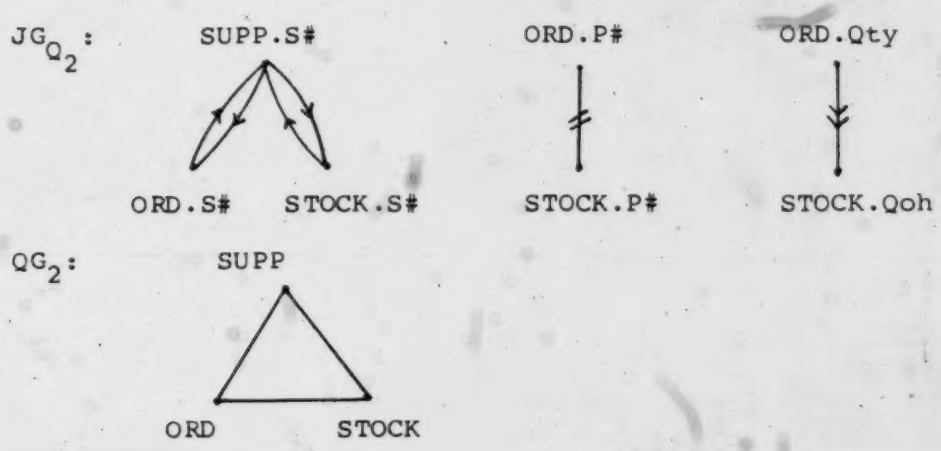
For equi-join queries, it was shown in [3] that if the query graph of Q is a tree then Q can be answered* by semi-joins. This result has a direct extension to queries involving join clauses with $\{\geq, =, >, \neq\}$. In addition, if the query graph of Q is cyclic, it may be possible to transform Q into an equivalent query such that the equivalent query has a tree query graph.

A query, Q, is called a tree query if either Q itself or a query equivalent to Q has a tree query graph. All other queries are called cyclic queries [3]. Thus a query can be one of the two types: a tree or a cyclic query. The set of all tree queries are denoted by TQ and all cyclic queries by CQ. Tree and cyclic queries are illustrated in Fig. 2.2.

---

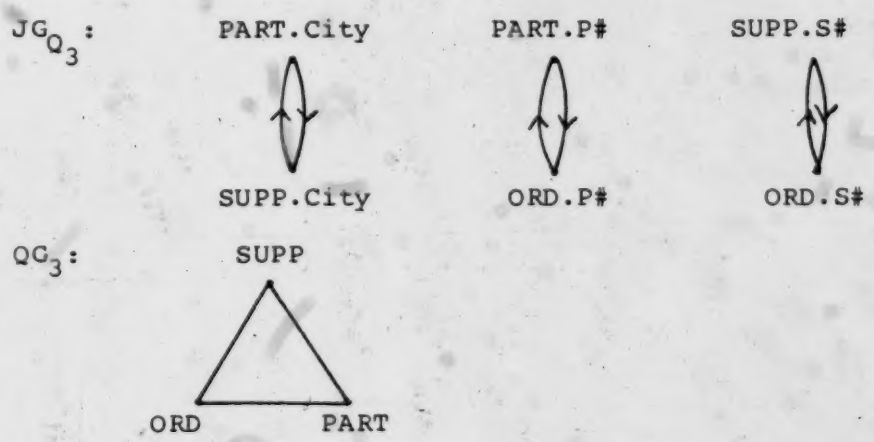
*A query Q without a target list is said to be answered if for every relation referenced in Q, the set of tuples satisfying (the qualification of) Q is found. If Q has a target list then it is sufficient to find only those tuples that are described in the target list and satisfy the qualification of Q. Hence a query with a target list can also be answered by semi-joins if its query graph is a tree.

Q_2: (SUPP.S# = ORD.S#).and.(SUPP.S# = STOCK.S#)
.and.(ORD.P# ≠ STOCK.P#).and.(ORD.Qty > STOCK.Qoh)



(a) A Tree Query (Equivalent to Q_1 in Fig. 2.2) with a Cyclic Query Graph.

Q_3: (PART.City = SUPP.City).and.(PART.P# =ORD.P#)
.and.(SUPP.S# = ORD.S#)



(b) A Cyclic Query.

Figure 2.2. Tree and Cyclic Queries.

CHAPTER 3

DETERMINING TREE QUERY MEMBERSHIP OF A DISTRIBUTED QUERY

3.1 Introduction

Clearly, if the query graph of a given query is a tree then the query is a tree query. However, a query with a cyclic query graph may also be a tree query (e.g. query Q_2 in Fig. 2.2). Thus, a procedure is required to determine the type of a query in general. For a query Q, a procedure which enumerates all the queries equivalent to Q and checks whether any of their query graphs is a tree, would be sufficient to determine the type of Q. However, such an exhaustive enumeration can be avoided by using a canonical representation of a query, called a reduced join graph.

In this chapter, first the reduced join graph is introduced then an efficient tree query membership algorithm is given for conjunctive queries with relational operators $\{>, \geq, =, \neq, <, \leq\}$. For tree queries, the algorithm also produces an equivalent query together with its tree query

graph so that the sequence of semi-joins to evaluate the query can immediately be constructed. An implementation of the algorithm and its time and space complexities are given.

3.2 Reduced Join Graph

Intuitively, the reduced join graph of a query Q is a join graph representing an equivalent query which has the fewest number of join clauses among all queries equivalent to Q. Such a join graph is obtained by first grouping vertices into equivalence classes (where any two vertices are in an equivalence class iff there is a cycle in the join graph incident with both vertices), and transforming the arcs within each equivalence class into a set of arcs representing only equi-join clauses; and then from the set of arcs incident with vertices in different equivalence classes, removing all redundant arcs and replacing the dominated arcs. In somewhat more detail, the construction of the canonical join graph consists of the following two steps.

(i) The construction of equivalence classes and corresponding spanning trees: Let C be a connected component in JG_Q. As discussed previously (Section 2.2), a cycle in C represents a conjunction of equi-join clauses of the form (u=v) for every pair of vertices, u, v, incident with the cycle. Thus vertices in a connected component can be partitioned into vertex equivalence classes such that any

two vertices are in the same equivalence class iff there is a cycle incident with both vertices, and a vertex forms an equivalence class by itself iff it is not incident with any cycle in C. Consequently, the set of arcs within each multimember vertex equivalence class can be replaced by a spanning tree of the vertices in the equivalence class, where each "edge" (u,v) of the spanning tree consists of both arcs $(u,v)_1$ and $(v,u)_1$. Since there is no self loop in C, it is clear that there is no arc within a single member vertex equivalence class.

(ii) The removal of redundant arcs: This proceeds by the construction of a condensed acyclic graph $AC(V_1,A_1)$ for each connected component $C(V_2,A_2)$ in the original join graph. Each vertex equivalence class in C is condensed to a vertex in V_1. The arcs in AC are determined by the arcs in C, as follows. For any two equivalence classes E_i, E_j, $i\neq j$, let $S_{i,j}$ be the set of arcs in C from vertices in E_i to vertices in E_j, i.e.,

$$S_{i,j} = \{ (u,v)_k \mid (u,v)_k \in A_2,\ u\in E_i,\ v\in E_j \text{ and } 1\leq k\leq 3 \}.$$

If $S_{i,j}\neq\emptyset$ then there is exactly one arc in AC of the form (E_i,E_j); otherwise there is no arc from E_i to E_j in AC. Suppose $S_{i,j}\neq\emptyset$. The arc in AC from E_i to E_j is a type-1 arc if all the arcs in $S_{i,j}$ are type-1 arcs; it is a type-3 arc if all the arcs in $S_{i,j}$ are type-3 arcs; and it is a type-2 arc otherwise. The resulting condensed graph, AC, is acyclic since any cycle in C must be within an equivalence

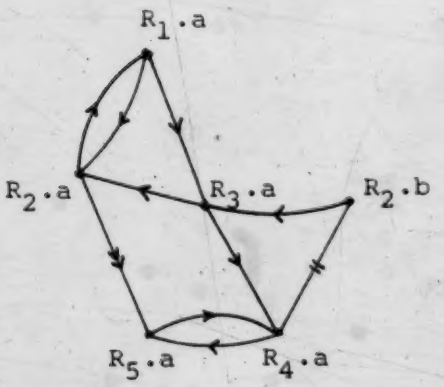
class, which in turn is a single vertex in AC. The arcs in the transitive closure of AC are successively examined, in any order, and those which are implied by transitivity are removed. The resulting graph, denoted AC^t, is called the transitive reduction (see Appendix A) of the acyclic graph AC. (Notice that the transitive reduction in our context is an extension of [1] for directed graphs containing three types of arcs. Appendix A formalizes the construction of the transitive reduction.) Finally, each vertex in V_1 is expanded by replacing it with a spanning tree of the vertices in the equivalence class corresponding to that vertex. The connected component of a reduced join graph, corresponding to the connected component C of JG_Q, is obtained by expanding every vertex in AC^t in this manner.

Thus, a reduced join graph for Q is obtained by performing steps (i) and (ii) for every connected component in JG_Q (see Fig. 3.1).

Since the construction of the reduced join graph is based on the transitive reduction of an acyclic graph, we will utilize the properties of the transitive reduction in the following analysis. Those properties are stated in lemma 3.1 below.

$Q: (R_1.a = R_2.a).\underline{and}.(R_1.a \geq R_3.a).\underline{and}.(R_3.a \geq R_2.a)$
$(R_2.a > R_5.a).\underline{and}.(R_4.a = R_5.a).\underline{and}.(R_3.a \geq R_4.a)$
$(R_2.b \geq R_3.a).\underline{and}.(R_2.b \neq R_4.a)$

(a) A Query Q

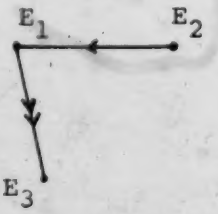


(b) A Join Graph JG

$E_1 = \{R_1.a, R_2.a, R_3.a\}$
$E_2 = \{R_2.b\}$
$E_3 = \{R_4.a, R_5.a\}$

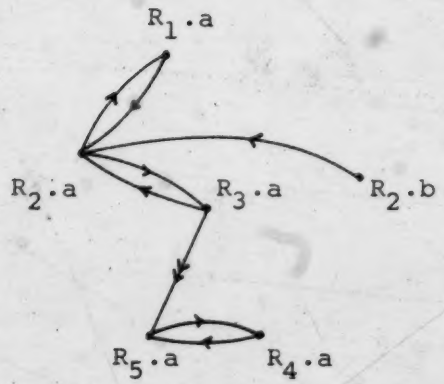
(c) Vertex Equivalence Classes

(d) Acyclic Graph of JG

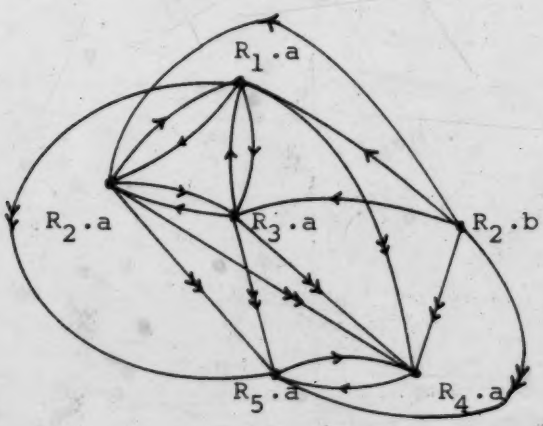


(e) Transitive Reduction of the Acyclic Graph in (d)

Figure 3.1. Construction of Reduced Join Graph



(f) A reduced join graph for JG



(g) Transitive Closure of the Join Graph of Q

Figure 3.1. Construction of Reduced Join Graph (cont'd).

Lemma 3.1: The transitive reduction, G^t, of a finite acyclic join graph G satisfies

(i) $(G^t)^T = G^T$, and

(ii) If $H^T = G^T$ then for any arc $(u,v)_i$ in G^t there is an arc $(u,v)_j$ in H, $1 \leq i,j \leq 3$.

Proof: Directly follows from Proposition A.1 in Appendix A. #

Thus, for a given acyclic join graph, G, any join graph having the same transitive closure as G contains at least as many arcs as the transitive reduction, G^t, of G. Moreover, if the types of the arcs are ignored then $G^t \subseteq H$ for any H satisfying $H^T = G^T$.

The reduced join graph obtained from a join graph JG_Q is not unique unless every vertex equivalence class in JG_Q is a single member equivalence class. For example, consider any two reduced join graphs JG_1 and JG_2 obtained from a given join graph JG_Q. Clearly JG_1 and JG_2 have the same set of vertex equivalence classes. The corresponding vertex equivalence classes in JG_1 and JG_2 have the same set of vertices but the spanning trees of vertices in the same equivalence class may not be the same. There is an arc $(u_1,v_1)_k$, $1 \leq k \leq 3$, in JG_1 where $u_1 \in E_i$, $v_1 \in E_j$, $i \neq j$, iff there is a corresponding arc $(u_2,v_2)_k$ in JG_2 satisfying $u_2 \in E_i$, $v_2 \in E_j$. However these arcs may differ in their end vertices, i.e., $u_1 \neq u_2$ or $v_1 \neq v_2$ is possible. The following lemma shows that any reduced join graph of a query Q represents a query

which is equivalent to Q.

Lemma 3.2: If JG is a reduced join graph for Q then $JG^T = JG_Q^T$.

Proof: By construction, JG and JG_Q have the same set of vertex equivalence classes, and the condensed acyclic graph of JG is the transitive reduction of the condensed acyclic graph of JG_Q. Thus, from Lemma 3.1, the condensed acylic graphs of the join graphs JG and JG_Q have the same transitive closure. Since it is straightforward to show that any two join graphs have the same transitive closure if they have the same vertex equivalence classes and the transitive closure of their condensed acyclic graphs are the same, it follows that $JG^T = JG_Q^T$. #

For any query Q, let AJ(Q) denote the set of all join graphs having the same transitive closure as that of the join graph of Q, i.e. $AJ(Q) = \{ JG \mid JG^T = JG_Q^T \}$. Clearly, AJ(Q) represents all queries, including Q itself, which are equivalent to Q. By definition, if there is a join graph in AJ(Q) whose query graph is a tree then Q is a tree query, otherwise it is a cyclic query. Let RJ(Q) denote the set of all reduced join graphs for Q. From lemma 3.2, $RJ(Q) \subseteq AJ(Q)$. The following lemma shows that in order to determine the type of a query Q it suffices to consider only the join graphs in the subset RJ(Q). This justifies that a reduced join graph is a canonical representation of a query.

Lemma 3.3: $Q \in TQ$ iff there is a join graph JG in RJ(Q) such that the query graph of JG is a tree.

Proof: The if-part follows immediately since $RJ(Q) \subseteq AJ(Q)$. For the only-if-part, let $JG_\emptyset$ be a join graph in AJ(Q) whose query graph, $QG_\emptyset$, is a tree. Then a reduced join graph JG can be constructed from $JG_\emptyset$ such that the set of edges in the query graph, QG, of JG is a subset of that in $QG_\emptyset$. Both QG and $QG_\emptyset$ have the same set of vertices and since $QG_\emptyset$ is a tree, from lemmas 3.2 and 2.2, QG is connected and is also a tree. #

In order to determine the type of a query Q, a reduced join graph obtained from JG_Q can be utilized with no loss of generality as is shown by the preceding lemma. However some of the vertices and arcs in a reduced join graph can be eliminated by local processing. Consider a spanning tree corresponding to a vertex equivalence class in a reduced join graph. Each edge in the spanning tree represents an equi-join clause. Equi-join clauses between the attributes of the same relation can be eliminated as a result of local processing. Thus, if there is a join attribute of a relation in a vertex equivalence class then the other attributes of the same relation are assumed to be absent from that vertex equivalence class.

For simplicity, we say that a connected component, C_K, contains R_i if there is at least one attribute of R_i in C_K. Similarly, an equivalence class contains R_i if an attribute

of R_i is in that equivalence class. We also say that R_i is adjacent to (from) R_j, if the arc $(R_i.a, R_j.b)_t$ ($(R_j.b, R_i.a)_t$) is in C_k where $R_i.a$ and $R_j.b$ are some attributes of R_i and R_j respectively, and $1 \leq t \leq 3$. It is assumed that each connected component in JG contains at least two relations, because otherwise all the join clauses in the connected component could be performed by local processing, and hence that connected component could be eliminated from JG. The next section is concerned with determining the type of a query Q using a reduced join graph for Q.

3.3 A General Tree-query Membership Algorithm

The approach we use in determining the type of a given query is similar to the one described in [29]. There, the relations and the corresponding clauses in a query are successively examined and those whose removal do not influence the type of the query are eliminated. If all the relations and the join clauses can be eliminated in this manner (resulting in a null join graph and a null query graph) then the final query and, more importantly, the original query are tree queries. On the other hand, if some relations and join clauses remain then both the resulting and the original queries are cyclic queries.

The procedure here, in somewhat more detail, is as follows. Consider a reduced join graph, JG, of a query after

all the local processing described in the preceding section is performed. Let $E_1,\ldots,E_p$ denote the vertex equivalence classes in JG. Associated with each relation, R_i, two sets M_i and S_i are defined as the set of multimember equivalence classes and the set of single member equivalence classes containing R_i respectively. That is, $M_i = \{E_t \mid E_t \text{ contains } R_i \text{ and } |E_t|>1\}$ and $S_i = \{E_t \mid E_t \text{ contains } R_i \text{ and } |E_t|=1\}$, $1\leq i\leq n$. Let R_i and R_j be any two relations involved in Q, $i\neq j$, $1 \leq i,j \leq n$. R_i is said to be <u>superseded</u> by R_j if

(1) $M_i \subseteq M_j$, and

(2) For each $E_t \in S_i$, every vertex E_k that is adjacent to or from E_t in the condensed acyclic graph of JG satisfies $E_k \in S_i \cup M_j \cup S_j$.

To illustrate the two conditions above, suppose R_i is superseded by R_j. Consider any joining attribute, say $R_i.a$, of R_i. $R_i.a$ is either in a multimember equivalence class or in a single member equivalence class. In the first case, that multimember equivalence class also contains R_j (from condition 1). In the latter case, every vertex v that is adjacent to or from $R_i.a$ is either in an equivalence class which also contains R_j or v is another attribute of R_i and it is in a single vertex equivalence class (from condition 2). Clearly, the same conditions are satisfied by every attribute of R_i.

If R_i is superseded by R_j then it will be shown (lemmas 3.5 and 3.6) that in order to determine the type of Q it

suffices to consider only a subset of all reduced join graphs for Q since $Q \in TQ$ iff there is a join graph in the subset whose query graph is a tree. Furthermore, every join graph in the subset has the property that all the arcs incident with attributes of R_i in such a join graph correspond to a single edge, (R_i,R_j), in the query graph. That is, the query graph of any join graph in the subset has exactly one edge incident on R_i. Consequently, any cycle (if one exists) in such a query graph can not be incident with R_i, and hence removing R_i will not affect the type of Q (prop. 3.1). R_i is removed from Q by eliminating all the attributes of R_i and the arcs incident with them from a join graph in the subset. In the remaining join graph there may be some connected components each containing a single relation. Those connected components can be removed as they do not contribute any edges to the query graph and therefore have no influence on the type of Q. However the removal of R_i and the subsequent removals of single relation connected components (if they exist) may cause some relations to become superseded by some other relations. The process of elimination continues until either all relations and the corresponding arcs are removed (giving a null join graph) or some relations remain such that for every pair of relations R_i, R_j in the remaining set, R_i is not superseded by R_j and each connected component contains at least two relations. In the former case, Q is a tree query since a null query graph is a tree query graph and each removal preserves the type of

the query. Furthermore, the conjunction of join clauses corresponding to the set of removed arcs is a query equivalent to Q and the query graph of this query is a tree (prop. 3.1). In the latter case, proposition 3.2 shows that the resulting query must be a cyclic query which in turn implies that Q is also a cyclic query.

We now proceed to show the above statements. Let RJ(Q) be the set of all reduced join graphs for Q. Suppose R_i is superseded by R_j, $i \neq j$, $1 \leq i,j \leq n$. A mapping M is defined which maps connected components in join graphs in RJ(Q) to connected components in join graphs of a subset of RJ(Q), denoted by $\bar{RJ}(Q)$, such that the query graph of any join graph in $\bar{RJ}(Q)$ contains the edge (R_i,R_j) but does not contain any edge (R_i,R_k), $k \neq j$. M leaves all arcs which are not incident with R_i unchanged, and replaces certain arcs incident with R_i by some arcs incident with R_j. The mapping M has the following properties:

(a) If E_k is a vertex equivalence class, so is $M(E_k)$,

(b) If there is an arc between two vertex equivalence classes, M either leaves the arc unchanged or replaces it by another arc of the same type between the same two equivalence classes.

More precisely, M is described as follows. Let JG be a join graph in RJ(Q) and C be a connected component in JG. C contains one or more vertex equivalence classes. We first describe how M modifies the arcs within equivalence classes. Let T_k denote a spanning tree corresponding to a vertex equivalence class E_k in C, such that an "edge" (u,v) in T_k denotes the two arcs (u,v) and (v,u). E_k is one of the following types:

(1) it does not contain R_i,

(2) it contains R_i, R_j and the edge $(R_i.a, R_j.b)$ is in T_k,

(3) it contains R_i, R_j but the edge $(R_i.a, R_j.b)$ is not in T_k, or

(4) it contains R_i but not R_j,

where $R_i.a$ and $R_j.b$ denote the attributes of R_i and R_j in E_k respectively. M does not change the equivalence classes of type (1). If E_k is of type (2), edges of the form $(R_i.a, v)$, $v \in E_k$, $v \neq R_j.b$, are mapped to $(R_j.b, v)$ while other edges of T_k are unchanged. If E_k is of type (3), let the edges incident with $R_i.a$ in T_k be $(R_i.a,v_1),\ldots,(R_i.a,v_s)$, $s \geq 1$. T_k is a tree, so there is a unique path in T_k from $R_i.a$ to $R_j.b$ passing exactly one vertex in $\{ v_t \mid 1 \leq t \leq s \}$. Without loss of generality let v_1 be that vertex. Then M maps the edge $(R_i.a, v_1)$ to $(R_i.a, R_j.b)$ and the edges $(R_i.a, v_t)$, $2 \leq t \leq s$ are mapped to $(R_j.b, v_t)$ in T_k. Equivalence classes of type (4) are not changed by M. (Notice that, such an equivalence class consists of only one vertex, namely $R_i.a$, since R_i is superseded by R_j.)

The arcs incident with vertices in different vertex equivalence classes in C are mapped as follows. Consider an arc $(u,v)_t$, $1 \leq t \leq 3$, in C such that u and v are in different equivalence classes, $u \in E_m$, $v \in E_k$ and $m \neq k$. If neither u nor v is an attribute of R_i then the arc $(u,v)_t$ is not changed by M. Otherwise, without loss of generality let u be $R_i.a$. E_m can not be a vertex equivalence class of type (1) since it contains R_i. If E_m is of types (2) or (3) then the arc $(R_i.a, v)_t$ is mapped to the arc $(R_j.b, v)_t$, where $R_j.b$ is the attribute of R_j in E_m. Otherwise E_m is of type (4) (i.e. a single vertex equivalence class containing only $R_i.a$) and since R_i is superseded by R_j there are three possible cases for v:

(i) v is an attribute of R_i in E_k and $|E_k|=1$,

(ii) v is an attribute of R_j in E_k,

(iii) v is an attribute of neither R_i nor R_j, but E_k contains R_j.

In cases (i) and (ii) the arc $(R_i.a, v)_t$ is left unchanged by M. For case (iii), let $R_j.b$ denote the attribute of R_j in E_k. M maps the arc $(R_i.a, v)_t$ to the arc $(R_i.a, R_j.b)_t$.

Lemma 3.4: If C is a connected component in $JG_\emptyset$, $JG_\emptyset \in RJ(Q)$, then C and M(C) have the same vertex equivalence classes and their condensed acyclic graphs are the same.

Proof: Follows from the construction of M which possesses properties (a) and (b). #

Since $JG_\emptyset$ is a reduced join graph, it follows from Lemma 3.4, that $M(JG_\emptyset)$ is also a reduced join graph for Q. Consequently, $\bar{RJ}(Q) \subseteq RJ(Q)$ (from Lemma 3.2), and both $JG_\emptyset$ and $M(JG_\emptyset)$ represent queries which are equivalent to Q (from Lemmas 3.2 and 2.1). The following two lemmas show that the query graph of a join graph in $RJ(Q)$ is a tree iff the query graph of the corresponding join graph in $\bar{RJ}(Q)$ is a tree. Thus, in order to determine the type of a query Q it is sufficient to consider only the join graphs in $\bar{RJ}(Q)$. In Lemmas 3.5 and 3.6, below, R_i and R_j are any two relations such that R_i is superseded by R_j.

Lemma 3.5: If there is a join graph JG_1 in $\bar{RJ}(Q)$ such that the query graph of JG_1 is a tree then there is a join graph in $RJ(Q)$ whose query graph is also a tree.

Proof: Since $\bar{RJ}(Q) \subseteq RJ(Q)$, $JG_1 \in RJ(Q)$. #

Lemma 3.6: Let $QG_\emptyset$ be the query graph of a join graph $JG_\emptyset \in RJ(Q)$. If $QG_\emptyset$ is a tree then the query graph of $M(JG_\emptyset)$ is also a tree.

Proof: Let JG_1 denote $M(JG_\emptyset)$ and QG_1 denote the query graph of JG_1. We now show that QG_1 is a tree by demonstrating that QG_1 is connected and has the same number of edges as $QG_\emptyset$. From Lemma 3.4, $JG_1^T = JG_\emptyset^T$. Since $QG_\emptyset$ is connected, QG_1 is also connected, by Lemma 2.2. $QG_\emptyset$ may or may not have the

edge (R_i, R_j).

Case 1: QG_0 contains the edge (R_i, R_j). Since M leaves all edges not incident with R_i unchanged, any edge which is in QG_0, but not in QG_1, must be incident on R_i. Let (R_i, R_t), $t \neq j$, $t \neq i$, be such an edge. Then there exists at least one arc in QG_0 that is incident with some attribute of R_i, $R_i.a$, and some attribute of R_t, $R_t.b$.

If $R_i.a$ and $R_t.a$ are in the same equivalence class, say E_m, then E_m must also contain an attribute of R_j, say $R_j.c$, since R_i is superseded by R_j. Moreover, the spanning tree T_m (representing E_m) must contain the edge $(R_i.a, R_j.c)$ since otherwise there would be an alternate path in QG_0 from R_i to R_j via some other vertex, contradicting that QG_0 is a tree. Thus, E_m is an equivalence class of type (2), so the edge $(R_i.a, R_t.b)$ is mapped to $(R_j.c, R_t.b)$ by M. Thus, QG_1 contains the edge (R_j, R_t).

Now suppose $R_i.a$ and $R_t.b$ are in different vertex equivalence classes, $R_i.a \in E_m$, $R_t.b \in E_k$, $m \neq k$, and let the arc in between be $(R_i.a, R_t.b)_s$, $1 \leq s \leq 3$. If E_m is not a single member equivalence class then an attribute of R_j, say $R_j.d$, must exist in E_m since R_i is superseded by R_j. M maps the arc $(R_i.a, R_t.b)_s$ to $(R_j.d, R_t.b)_s$. If E_m is a single member equivalence class then E_k contains an attribute of R_j, say $R_j.e$. (Note that E_k can not be a single member equivalence class containing only R_i since the edge under consideration is $(R_i.a, R_t.b)_s$, $t \neq i$, $t \neq j$.) Since QG_0 is a tree, E_k must also contain another attribute of R_i, say $R_i.f$, and T_k must

contain the unique path $(R_j.e, R_i.f)$, $(R_i.f, R_t.b)$ from $R_j.e$ to $R_t.b$. M maps the arc $(R_i.a, R_t.b)_s$ to $(R_i.a, R_j.e)_s$ and it maps the edge $(R_i.t, R_t.b)$ in T_K to $(R_j.e, R_t.b)$. Thus, in either situation QG_1 contains the edge (R_j, R_t). $QG_\emptyset$ can not contain (R_j, R_t) since it is a tree. Therefore QG_1, in comparison with $QG_\emptyset$, loses one edge (R_i, R_t) but gains (R_j, R_t). Since the same argument is applicable for every edge (R_i, R_t), $t \neq i$, $t \neq j$, QG_1 has the same number of edges as $QG_\emptyset$.

Case 2: $QG_\emptyset$ does not contain the edge (R_i, R_j). Since $QG_\emptyset$ is a tree there is a unique path in $QG_\emptyset$ from R_i to R_j. Let this path be $e=(R_i, R_{k_1}, \ldots, R_{k_p}, R_j)$, $p \geq 1$. Consider any connected component C in $JG_\emptyset$ containing R_i. C also contains R_j, and hence there is a path in $QG_\emptyset$ from R_i to R_j which is mapped by the arcs in C. This path must be the same as e since $QG_\emptyset$ is a tree. The remaining arguments are similar to those of case 1. #

Proposition 3.1: Let Q be a query such that R_i is superseded by R_j, for some $i \neq j$, $1 \leq i, j \leq n$. $Q \in TQ$ iff there exists a tree query graph corresponding to a join graph in $\bar{R}\bar{J}(Q)$. Furthermore any cycle in the query graph corresponding to a join graph in $\bar{R}\bar{J}(Q)$ cannot be incident on R_i.

Proof: Follows from Lemmas 3.5, 3.6 and the fact that R_i has only one edge incident on it in the query graph corresponding to any join graph in $\bar{R}\bar{J}(Q)$. #

Thus no existing cycle in the query graph can be

incident on R_i. As a consequence, the removal of R_i does not effect the query type. Observe that "superseding" is independent of which reduced join graph is considered since all the reduced join graphs have the same vertex equivalence classes and their condensed acyclic graphs are the same.

We now proceed with the removal of R_i from a join graph JG, $JG \in \overline{RJ}(Q)$, and show that the join graph remaining after the removal of R_i is in canonical form. The relation R_i is removed by eliminating all the attributes of R_i and the arcs incident on them from JG. The removal of R_i may cause some connected components in JG to consist of attributes of a single relation. Such connected components are also eliminated since those connected components do not contribute any edges to the query graph, and therefore have no influence on the type of Q. In essence, a mapping ER_i is defined where $ER_i(JG)$ is the join graph resulting after the elimination of R_i from JG, where $JG \in \overline{RJ}(Q)$. (Note that if JG contains R_i and R_j but no other relation then $ER_i(JG)$ is a null join graph.) From Proposition 3.1, it is clear that for any two join graphs JG_1 and JG_2 in $\overline{RJ}(Q)$, the queries represented by $ER_i(JG_1)$ and $ER_i(JG_2)$ are equivalent. Moreover, the following lemma shows that $ER_i(JG)$ is a reduced join graph for any $JG \in \overline{RJ}(Q)$.

Lemma 3.7: Let JG' be the join graph resulting after the removal of a superseded relation R_i from JG, i.e. $JG' = ER_i(JG)$, where $JG \in \bar{\bar{RJ}}(Q)$. Then JG' is a reduced join graph.

Proof: Suppose JG' is not null since otherwise the result is trivially true. By construction, for each equivalence class E'_k in JG', JG contains a corresponding equivalence class E_k such that $E'_k \subseteq E_k$. From proposition 3.1, each equivalence class E'_k in JG' is represented by a spanning tree of the vertices in E'_k. Moreover, JG' contains an arc $(u,v)_s$, $1 \leq s \leq 3$, where $u \in E'_k$, $v \in E'_m$, $k \neq m$, only if JG contains the same arc, $(u,v)_s$, satisfying $u \in E_k$, $v \in E_m$. Since JG is a reduced join graph it does not contain any redundant arc incident with vertices in different equivalence classes. This shows that JG' also does not contain any redundant arc. Therefore JG' is also a reduced join graph. #

Thus the removal of the superseded relation R_i from a join graph in $\bar{\bar{RJ}}(Q)$ results in a reduced join graph. Let $RJ(Q')$ denote the set of all such join graphs, i.e., $RJ(Q') = \{JG' \mid JG' = ER_i(JG)$ and $JG \in \bar{\bar{RJ}}(Q)\}$, and Q' be the query represented by a join graph in $RJ(Q')$. It is evident that $RJ(Q')$ is the set all reduced join graphs for Q'. Consequently, the type of the original query Q can be determined from a join graph resulting after the removal of R_i from any join graph in $\bar{\bar{RJ}}(Q)$.

Finally proposition 3.2 shows that if the elimination process leaves some relations then the query must be a cyclic query.

Proposition 3.2: Let Q be a query involving relations $\{R_1,\ldots,R_n\}$ such that R_i is not superseded by R_j for any $i\neq j$, $1 \leq i,j \leq n$, and each connected component contains at least two relations. Then $Q \in CQ$.

Proof: Let QG be the query graph of any join graph, say JG, in RJ(Q). From Lemma 3.3, it suffices to show that QG is cyclic. Consider R_i in QG. An attribute of R_i is adjacent to or from some vertex which is an attribute of another relation, say R_j, in a connected component in JG. Thus, R_i and R_j are also adjacent in QG. Since R_i is not superseded by R_j, at least one of the two conditions for R_i to be superseded by R_j does not hold.

Case 1: $M_i \not\subseteq M_j$. Then there exists a multimember equivalence class E_k such that $E_k \in M_i$ and $E_k \notin M_j$. Let $R_i.a$ be the attribute of R_i in E_k. Since $|E_k| \geq 2$, there is at least one other vertex, $R_t.b$, $t\neq i$, $t\neq j$, in E_k such that $R_t.b$ is adjacent to $R_i.a$ in the spanning tree representing E_k. Thus R_t, $t\neq i$, $t\neq j$, is also adjacent to R_i in QG showing that the degree of R_i in QG is at least 2.

Case 2: In the condensed acyclic graph of JG there is an arc incident with E_k and E_m, where $E_k \in S_i$ and $E_m \notin S_i \cup M_j \cup S_j$. Let $R_i.a$ be the attribute of R_i in E_k. Then there exists at least one vertex, $R_t.b$, $t\neq i$, $t\neq j$, in

E_m such that JG contains an arc incident with $R_i.a$ and $R_t.b$. Thus, R_i and R_t are also adjacent in QG. This shows that the degree of R_i is at least 2.

Since the above argument is applicable to every vertex in QG, each vertex in QG has degree ≥ 2, and therefore [12] QG is cyclic. #

Below we present an algorithm which determines the type of a given query Q by successively examining each pair of relations R_i, R_j in Q and removing R_i if R_i is superseded by R_j, $i \neq j$, $1 \leq i,j \leq n$. For any tree-query, Q, the algorithm produces an equivalent query whose query graph is a tree, and outputs the equivalent query together with its tree query graph. The input query Q is free of join clauses that can readily be eliminated by local processing, i.e., initially, each connected component in the join graph of Q contains two or more relations and there is at most one attribute of a relation within an equivalence class.

Algorithm 3.1:

Input: Query Q

Output: If Q∈TQ then "Q∈TQ", an equivalent query, Q^*, and tree query graph, T, of Q^*; else "Q∈CQ".

1) Construct JG_Q from Q. Initialize T to be an empty set and Q^* be a null query.

2) Construct the vertex equivalence classes $E_1,\ldots,E_p$, and the transitive reduction, G^t, of the condensed acyclic graph of JG_Q. For each relation R_i in Q, construct the sets M_i and S_i where

$M_i=\{E_t \mid E_t$ contains R_i and $|E_t|>1\}$,

$S_i=\{E_t \mid E_t$ contains R_i and $|E_t|=1\}$, $1\le i\le n$.

3) while (R_i is superseded by R_j) for some $i\neq j$, $1 \le i,j \le n$ do

/* (R_i is superseded by R_j) = True if $M_i \subset M_j$, and for every $E_t \in S_i$, each E_k that is adjacent to or from E_t in G^t satisfies $E_k \in S_i \cup M_j \cup S_j$, where $M_i \cup S_i \neq \emptyset$. */

Begin /* remove R_i */

(i) For every $E_t \in S_i \cup M_i$, remove the attribute of R_i from E_t and if $E_t \in S_i$ also remove E_t and all arcs incident with E_t from G^t. Set $M_i=S_i=\emptyset$. For each such removed attribute of R_i, say $R_i.K$, set $Q^* \leftarrow Q^*$.and.q where q is the join clause corresponding to the arc incident with $R_i.K$ in a join graph in $\overline{RJ}(Q)$.

(ii) Update M_j and S_j due to (i).

/* R_j is the only relation that may be affected by (i) since R_i is superseded by R_j. */

(iii) If $M_j=\emptyset$ and every $E_k \in S_j$ is such that there is no arc in G^t incident with E_k and E_m for $E_m \notin S_j$, then remove every $E_k \in S_j$ and the arcs incident with them (if they exist) from G^t. Set

$S_j = \emptyset$.

For each such removed arc a

set $Q^* \leftarrow Q^*$ and (the join clause corresponding to a).

/* Remove R_j, if each connected component containing R_j in the remaining join graph contains no other relation. */

(iv) Set $T \leftarrow T \cup \{(R_i, R_j)\}$.

end

4) If all the relations are eliminated from Q then return($Q \in TQ$, Q^* and T) else return($Q \in CQ$).

Clearly the algorithm terminates either when all the relations are eliminated from Q or when there is no R_i, R_j, $i \neq j$, among the relations remaining in Q, such that R_i is superseded by R_j. The correctness of the algorithm follows directly from propositions 3.1 and 3.2. Note that if the query graph of a query Q is a tree then the sequences of semi-joins to answer Q are determined directly by the tree query graph [3]. Thus, for any tree query Q the equivalent query Q^* produced by the algorithm can be utilized to answer Q by semi-joins since Q^* has a tree query graph (also produced by the algorithm) and the equivalent queries have the same answer.

3.4 An Implementation of the Algorithm

In step 1, an adjacency matrix is constructed for each connected component in JG_Q. Let $C_1,\ldots,C_m$, $m\geq 1$, be the connected components in JG_Q and n_i denote the number of vertices in C_i, $1 \leq i \leq m$. The vertices in the join graph can be numbered so that the vertices in the same connected component have successive indices. Let $R_s.a$ and $R_t.b$ be the i^{th} and j^{th} vertices respectively in a connected component C_k where $1 \leq i, j \leq n_k$, $1 \leq k \leq m$. The adjacency matrix A_k of a connected component C_k is an $(n_k x n_k)$ matrix where rows and columns represent the vertices in C_k and

$$A_k(i,j) = \begin{cases} 1 & \text{if } (R_s.a,\ R_t.b)_1 \text{ is in } C_k, \\ 2 & \text{if } (R_s.a,\ R_t.b)_2 \text{ is in } C_k, \\ 3 & \text{if } (R_s.a,\ R_t.b)_3 \text{ or} \\ & \quad (R_t.b,\ R_s.a)_3 \text{ is in } C_k, \\ \emptyset & \text{otherwise} \end{cases}$$

for $i\neq j$ and $A_k(i,i)=\emptyset$, $1 \leq i,j \leq n_k$, $1 \leq k \leq m$. Note that since $(R_s.a,\ R_t.b)_3$ is the same as $(R_t.b,\ R_s.a)_3$, $A_k(i,j)=A_k(j,i)=3$ if C_k contains either of the arcs $(R_s.a,\ R_t.b)_3$ or $(R_t.b,\ R_s.a)_3$. It is clear that constructing adjacency matrices for all the connected components of JG_Q takes at most $O(\sum_{i=1}^{m} n_i^2)$ time.

In step 2, the vertex equivalence classes and the condensed acyclic graph, G, of JG_Q can be constructed in

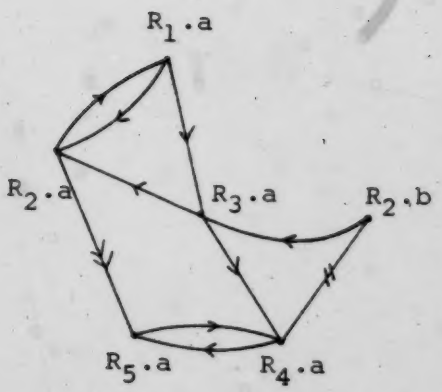
time proportional to $\sum_{i=1}^{m} n_i^2$ [15]. The transitive reduction, G^t, of the condensed acyclic graph G can be constructed in at most $O(\sum_{i=1}^{m} n_i^3)$ time (see Appendix A). Then the construction of sets M_i and S_i, for i=1,...,n, can be performed in time proportional to $\sum_{i=1}^{m} n_i$. Thus the complexity of step 2 is $O(\sum_{i=1}^{m} n_i^3)$. Since step 4 takes at most O(n) time, for the overall time complexity of the algorithm it remains to establish the time required by step 3.

In step 3, in order to determine whether $M_i \subseteq M_j$ the data structures and the procedure given in [29] can be employed as follows. (Note that a multimember equivalence class in this context corresponds to a connected component in [29].) For notational convenience, let $E_1,\ldots,E_t$, $1\leq t\leq p$ be the multimember equivalence classes, $E_{t+1},\ldots,E_p$ be the single member vertex equivalence classes in JG_Q, and $m_i=|E_i|$, $1\leq i\leq p$. As discussed previously (section 2), each multimember equivalence class contains at most one attribute of a given relation after local processing. Thus, the equivalence classes in JG_Q can be represented by an (nxp) binary matrix with a 1 in row i and column j if relation R_i is contained in equivalence class E_j. The nonzero entries of this matrix are represented by two sets of intersecting linked lists $\{LR_i \mid 1\leq i\leq n\}$ and $\{LE_j \mid 1\leq j\leq p\}$. The singly linked list LR_i represents the set $M_i \cup S_i$, while the doubly linked circular list LE_j represents the equivalence class E_j (see Figure 3.2). A vector of counts, CE, is used where

CE(i) is the number of relations in E_i, i.e., $CE(i)=|E_i|$.

The check for $M_i \subseteq M_j$, $1\leq i,j\leq n$, $M_i\neq\emptyset$, is facilitated by an (nxn) matrix, Count. Initially, $Count(i,j)=|M_i|$, $1\leq i,j\leq n$, and as the executions (to be described) proceed, $Count(i,j)=|M_i-M_j|$, $i\neq j$. Clearly, $Count(i,j)=0$ iff $M_i \subseteq M_j$. To find the set of M's contained in a given M_j, the linked list LR_j is traversed. This list intersects with the lists of equivalence classes, $\{LE_{j_s} \mid E_{j_s}$ contains $R_j\}$, containing R_j. Each occurrence of any relation R_u, $u\neq j$, in the list LE_{j_t} indicates that the equivalence class E_{j_t} also contains R_u. Thus, Count(u,j) should be decremented by the number of occurrences of R_u in such lists $\{LE_{j_s}\}$. This is accomplished by traversing the circular linked lists $\{LE_{j_s}\}$ while decrementing Count(u,j) for each occurrence of R_u, $u\neq j$, in those lists. Each node in LR_j denotes an occurrence of R_j in an equivalence class, E_s. While traversing the list LE_s, each node in the list is visited once and the traversal of LE_s resumes when the node in LR_j is reached (note that if $|E_s|=1$ then LE_s consists of one node, that is the node in LR_j). Thus the number of nodes visited in LE_s is the number of vertices in E_s. Since this process is carried out for each node in LR_j and for each list LR_j, the total time required for determining whether or not $M_i \subseteq M_j$, for all $i\neq j$, $1\leq i,j\leq n$, is $O(\sum_{k=1}^{p} m_k^2)$, where $m_k=|E_k|$, $1\leq k\leq p$.

In order to check for the second condition of supersedence in step 3, an (nxn) matrix, Scount, is



(a) A Join Graph JG

	E_1	E_2	E_3
R_1	1	0	0
R_2	1	1	0
R_3	1	0	0
R_4	0	0	1
R_5	0	0	1

(b) The Matrix Representation

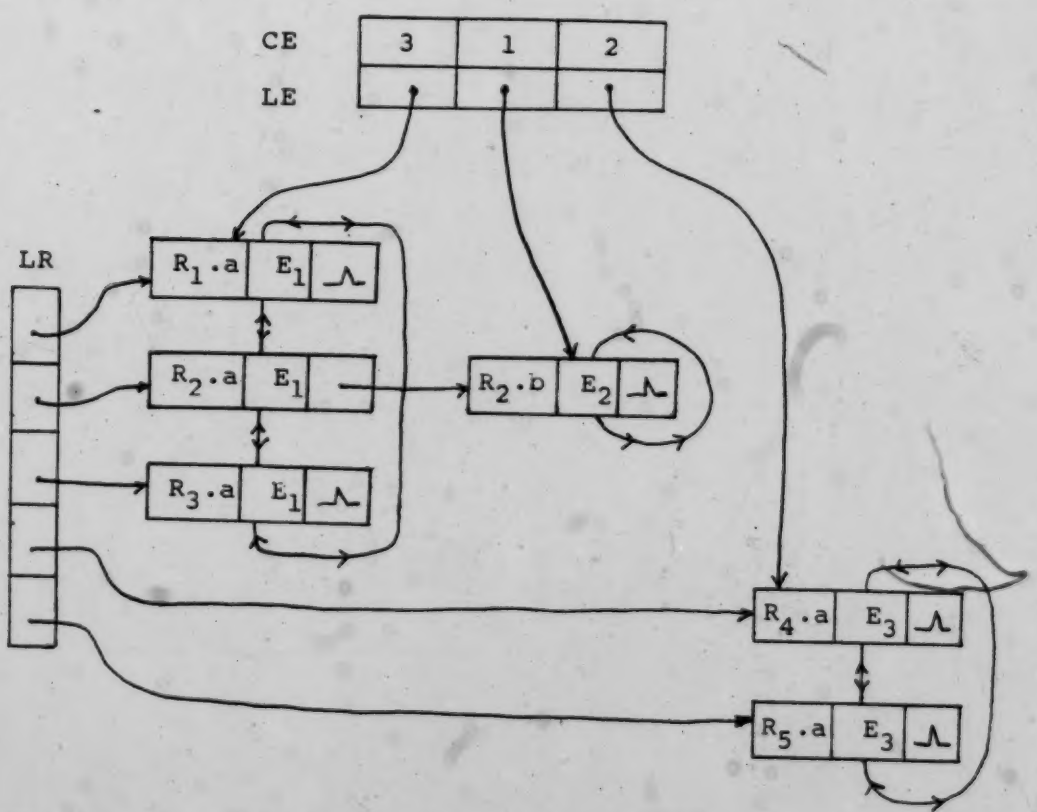


Figure 3.2. The Linked List Structure Used by the Algorithm.

utilized. All entries of Scount are initially 0. For a given relation R_i, the set of relations R_j, $i \neq j$, $1 \leq j \leq n$, such that the second condition for R_i to be superseded by R_j is satisfied, is determined as follows. For each E_k in S_i, every arc incident with E_k in G^t is traversed. Let E_u be a vertex adjacent to or from E_k in G^t. If $E_u \notin S_i$ then Scount(i,j) for all j satisfying $E_u \in M_j \cup S_j$, $j \neq i$, and Scount(i,i) are incremented by 1. (This is performed by traversing the linked list LE_u while incrementing Scount(i,i) by 1 and incrementing Scount(i,j) by 1 for each occurrence of R_j, $i \neq j$, in the list.) After repeating this process for every vertex in G^t that is adjacent to or from E_k, $E_k \in S_i$, and for every E_k in S_i, Scount(i,i)=Scount(i,j) for $i \neq j$ indicates that the second condition for R_i to be superseded by R_j is satisfied (i.e., R_i is superseded by R_j if $M_i \subseteq M_j$ and Scount(i,i)=Scount(i,j)). Clearly, the number of operations required for each E_k, $E_k \in S_i$, is proportional to the number of arcs in JG_Q^T incident with the attribute of R_i which is in E_k. Thus, all the relations, R_j, such that the second condition for R_i to be superseded by R_j is satisfied can be determined in time proportional to the number of arcs in JG_Q^T incident with those attributes of R_i which are in single vertex equivalence classes. This process is carried out for every relation R_i, $1 \leq i \leq n$. Hence, the total time required to check for the second condition of supersedence for every pair of relations R_i, R_j, $i \neq j$, $1 \leq i,j \leq n$, is at most $O(e_d)$ where e_d is the total number

of arcs in JG_Q^T incident with vertices that are in different equivalence classes.

If a relation, R_i, is superseded by another relation, R_j, then R_i is removed (step 3-i). The removal of R_i is performed by traversing the linked list LR_i once, and deleting every node in LR_i as it is encountered. Note that each node in LR_i is also a node in an intersecting list LE_k, where E_k is either in S_i or in M_i. The operations accompanied by the removal of R_i are described below in two cases.

Case 1: $E_k \in S_i$. Then the node in LR_i is the only node in the intersecting list LE_k. Thus, by the removal of the node in LR_i, the linked list LE_k becomes empty, and is deleted from the list structure. The vertex E_k and every arc incident with E_k are deleted from G^t. For each such deleted arc, $(E_k, E_u)_t$ or $(E_u, E_k)_t$, $1 \le t \le 3$, a corresponding join clause is stored in Q^* and the matrix Scount is updated as follows. Let $R_i.a$ be the attribute of R_i in E_k. Since R_i is superseded by R_j, E_u is either in S_i or in $M_j \cup S_j$. Let v denote the attribute of R_i in E_u if $E_u \in S_i$, and the attribute of R_j in E_u otherwise. The vertex v is determined by traversing the nodes in the linked list LE_u. Then the join clause corresponding to the arc $(E_k, E_u)_t$ ($(E_u, E_k)_t$) is $(R_i.a \ge v)$ $((R_i.a \le v))$ if t=1; $(R_i.a > v)$ $((R_i.a < v))$ if t=2; and $(R_i.a \ne v)$ $((R_i.a \ne v))$ otherwise. Observe that if $E_u \in S_j$ then (since $E_k \in S_i$) the arc, $(E_k, E_u)_t$ or $(E_u, E_k)_t$,

$1 \leq t \leq 3$, has been counted previously while determining the set of relations R_m, $m \neq j$, such that the second condition for R_j to be superseded by R_m is satisfied. Since the arc incident with E_k and E_u in G^t was just deleted due to the removal of R_i, if $E_u \in S_j$ then Scount(j,j) and Scount(j,i) should be decremented by 1. Thus, for every such arc, $(E_k, E_u)_t$ or $(E_u, E_k)_t$, deleted from G^t due to the removal of R_i, where $E_k \in S_i$, $1 \leq t \leq 3$, Scount(j,j) and Scount(j,i) are decremented by 1 if $E_u \in S_j$. The number of counts that has to be decremented in the matrix Scount for E_k is at most twice as many as the number of arcs in G^t incident with E_k, $E_k \in S_i$. For each arc in G^t incident with E_k, exactly one join clause is stored in Q^* and the time required to determine the join clause associated with such an arc $(E_k, E_u)_t$ or $(E_u, E_k)_t$ is at most proportional to the number of vertices in E_u. Thus, if $E_k \in S_i$ then the total time required by operations accompanied by the removal of the node from LR_i is proportional to the number of arcs in JG_Q^T incident with the attribute of R_i in E_k.

Case 2: $E_k \in M_i$. Then the node in LR_i is deleted from the intersecting (doubly linked) list LE_k. The count associated with LE_k, CE(k), is decremented by 1. Let A_i be the attribute of R_i in E_k. Since R_i is superseded by R_j, $M_i \subseteq M_j$, and hence E_k also contains an attribute of R_j. Let $R_j.b$ be the attribute of R_j in E_k, which can be identified by a traversal of the list LE_k. Then, an equi-join clause $(R_i.a = R_j.b)$ is stored in Q^*. Thus, if $E_k \in M_i$ then the

removal of the node from LR_i is performed in time proportional to the number of vertices in E_k.

Consequently, the time required for the removal of any node from LR_i is proportional to the number of arcs in JG_Q^T incident with the attribute of R_i corresponding to the removed node. Since every node in LR_i is removed in this manner, the total time required by the removal of R_i is proportional to the number of arcs in JG_Q^T incident with the attributes of R_i. All the relations are removed in the worst case, which takes O(e) time where e is the number of arcs in JG_Q^T.

The removal of a relation, R_i, may cause some multimember vertex equivalence classes to become singlemember equivalence classes. This may in turn cause some relations to become superseded by some other relations. Thus, the matrices Count and Scount have to be updated. Let $S=\{E_s \mid 1\leq s\leq p\}$ be the set of equivalence classes which become single member equivalence classes after the removal of R_i. The set S can be formed during the removal of R_i. Let R_j be the relation that superseded R_i, $i\neq j$. The relation contained in any equivalence class, E_s, in S is R_j because E_s previously contained an attribute of R_i and an attribute of R_j, and the attribute of R_i was just removed. Thus, after the removal of R_i, each equivalence class in S becomes a member of S_j while each such equivalence class was a member of M_j before the removal.

Since M_j becomes M_j-S, the matrix Count has to be updated accordingly (step 3-ii). That is, if there are f equivalence classes in S, $f \geq \emptyset$, then all the counts associated with M_j, i.e., Count(j,k), $1 \leq k \leq n$, are decremented by f. The number of counts that have to be decremented is at most n-1, since the number of relations to be removed until the algorithm terminates is at most n, the updating of the matrix count takes at most $O(n^2)$ time.

Similarly, since S_j becomes $S_j \cup S$ after the removal of R_i, all the counts associated with S_j, i.e., Scount(j,k), $1 \leq k \leq n$, have to be updated for the equivalence classes in S (step 3-ii). This proceeds by traversing all the arcs in G^t that are incident with E_s, for every $E_s \in S$. Let E_u be a vertex that is adjacent to or from E_s in G^t, $E_s \in S$. If $E_u \notin S_j$ then Scount(j,j) and Scount(j,k) for all k satisfying $E_u \in M_k \cup S_k$, $k \neq j$, are incremented by 1. This is repeated for every vertex, E_u, that is adjacent to or from E_s in G^t and for every E_s in S. Thus the updating of the matrix Scount requires twice as many operations as traversing those arcs JG_Q^T which are incident with an attribute of R_j in E_s and some other vertex in a different equivalence class, exactly once for every $E_s \in S$. However, each equivalence class in S was in M_j before the removal of R_i, and became a member of S_j after the removal of R_i. In the worst case, until the termination of the algorithm each equivalence class may become a single member equivalence

class only once. Thus the total time required to update the matrix Scount is at most $O(e_d)$, where e_d is the total number of arcs in JG_Q^T incident with vertices that are in different vertex equivalence classes.

As described above, if R_i is superseded by R_j, $i \neq j$, and is removed then the only relation that is affected by the removal is R_j. After the removal of R_i, R_j may become an "isolated" relation, (i.e., every connected component containing R_j in the remaining join graph may contain no other relation), or R_j may become superseded by some other relations. These cases are determined as follows. After updating the matrices Count and Scount due to the removal of R_i, Count(j,j)=$\emptyset$ and Scount(j,j)=$\emptyset$ indicates that $M_j=\emptyset$ and, either $S_j=\emptyset$ or for every $E_k \in S_j$, every vertex adjacent to or from E_k in G^t is also in S_j, i.e., R_j is an isolated relation (step 3-iii). On the other hand, Count(j,k)=$\emptyset$ Scount(j,j)=Scount(j,k), $k \neq j$, indicates that R_j has become superseded by R_j. In either case, R_j has to be removed as described previously. (Observe that if R_j is removed as being an isolated relation (step 3-iii) then all the join clauses stored in Q^* associated with the removal of R_j involve attributes of only R_j and no other relation.) The time required for the removal of R_j has already been taken into consideration. Associated with every relation R_i, which is removed as being superseded by some other relation R_j, $i \neq j$, an edge (R_i, R_j) is stored in T (step 3-iv). Since at

most n-1 such edges are stored in T, the overall complexity of step-3 is at most $O(\max(n^2,e))$ where e is the number of arcs in JG_Q^T.

Consequently, the worst case time complexity of the algorithm is $O(\max(n^2, \sum_{i=1}^{m} n_i^3))$ since $e \leq \sum_{i=1}^{m} n_i^2$, where n is the number of relations and n_i is the number of vertices in connected component C_i, $1 \leq i \leq m$.

For each vertex in a join graph of Q, there is a node in the linked lists $\{LR_i \mid 1 \leq i \leq n\} \cup \{LE_j \mid 1 \leq j \leq p\}$. Thus the storage requirement for the two sets of lists is proportional to the number of vertices in a join graph of Q, i.e., $O(\sum_{i=1}^{m} n_i)$. Clearly, the storage requirement for the adjacency matrices of all the connected components is $O(\sum_{i=1}^{m} n_i^2)$. Since the matrices Count and Scount take $O(n^2)$ storage the overall storage complexity of the algorithm is $O(\max(n^2, \sum_{i=1}^{m} n_i^2))$.

3.5 Summary

A canonical representation of a distributed relational query, called a reduced join graph is introduced. The query represented by a reduced join graph does not contain any redundant join clauses, and is equivalent to the original query. A conceptually simple algorithm for determining the type of such a query has been presented. For tree queries, the presented algorithm produces an equivalent query whose query graph is a tree, and outputs the equivalent query

together with its tree query graph. Tree queries can always be answered by semi-joins, which usually can be computed with much less data transmission than joins. Moreover, a sequence of semi-joins that can be used to answer a tree query simply corresponds to a traversal of a tree query graph of an equivalent query [3]. Thus, the presented algorithm can effectively be utilized to improve the performance of distributed query processing mechanisms.

CHAPTER 4

DISTRIBUTED QUERY OPTIMIZATION FOR TREE QUERIES

4.1 Introduction

This chapter presents an approach to find the optimum sequence of semi-joins for answering a class of tree queries. Queries have conjunctive equi-join qualifications, and the number of common joining attributes between any two relations is at most one. Considering only such queries has some simplifying consequences on the notation and the results of the previous two chapters, which are summarized below.

Since the qualification of a query Q is a conjunction of equi-join clauses, the join graph representing Q consists of only type-1 arcs (see Section 2.2). Consequently, such a query Q can be represented by an undirected join graph JG_Q where each equi-join clause is represented by an edge as opposed to two type-1 arcs in the representation given in Section 2.2 (see Example 4.1). The transitive closure JG_Q^T of a join graph JG_Q, in this context, is obtained by adding

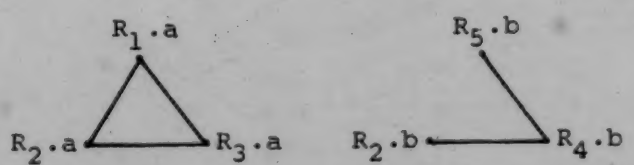
all possible edges to the join graph so that every connected component becomes maximally connected. For a given query Q, each connected component in the join graph JG_Q consists of a vertex equivalence class. Consequently, if there is an attribute of a relation in a connected component in JG_Q then the additional attributes of the same relation can be eliminated from the connected component by local processing (see Section 3.2). A reduced join graph of a query Q in this context is a spanning forest of the transitive closure JG_Q^T of the join graph.

Example 4.1: Consider the following equi-join query:

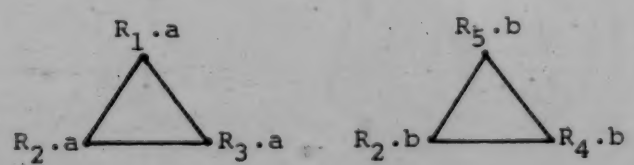
$$Q = (R_1.a = R_2.a).\underline{and}.(R_2.a = R_3.a).\underline{and}.(R_1.a = R_3.a)$$
$$.\underline{and}.(R_2.b = R_4.b).\underline{and}.(R_4.b = R_5.b)$$

The join graph JG_Q, transitive closure JG_Q^T and a reduced join graph for Q are given below.

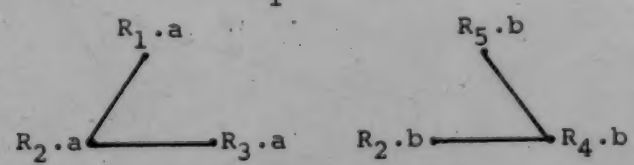
JG_Q:



JG_Q^T:



A reduced join graph JG_{Q_1}:



Let Q^* and Q_1 be the queries represented by JG_Q^T and JG_{Q_1} respectively. Then

$Q^* = (R_1.a = R_2.a).\underline{and}.(R_2.a = R_3.a).\underline{and}.(R_3.a = R_1.a)$
$.\underline{and}.(R_2.b = R_4.b).\underline{and}.(R_4.b = R_5.b).\underline{and}.(R_2.b = R_5.b)$

$Q_1 = (R_1.a = R_2.a).\underline{and}.(R_2.a = R_3.a)$
$.\underline{and}.(R_2.b = R_4.b).\underline{and}.(R_4.b = R_5.b)$

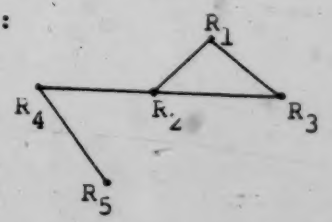
The queries Q, Q^* and Q_1 are equivalent queries since $JG_Q^T = JG_{Q_1}^T$. #

There is a one-to-one correspondence between the joining attributes of a relation and the connected components of the join graph. Hence the attributes of relations referenced in Q can be renamed so that the attributes which are the vertices of the same connected component in the join graph have the same name (see Example 4.1). Any two relations R_i, R_j are said to have a common attribute if a connected component contains attributes of both R_i and R_j (e.g. relations R_2, R_4, R_5 of Example 4.1 have attribute b in common). Since two relations R_i, R_j have at most one common attribute, $R_i \rightarrow R_j$ is sufficient to denote the semi-join without referencing the joining attributes.

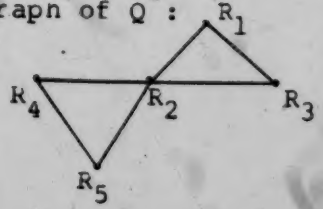
The query graph of a query is as described previously (Section 2.6). The following example illustrates the query graphs of three equivalent queries.

Example 4.2: Consider the query Q in example 4.1 and queries Q_1 and Q^* that are equivalent to Q. The query graphs of Q, Q_1 and Q^* are given below.

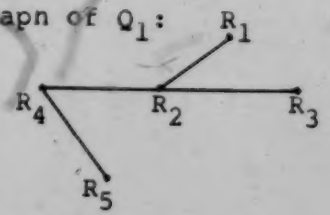
Query Graph of Q:



Query Graph of Q^*:



Query Graph of Q_1:



The following lemma shows that there is a one-to-one correspondence between the edges of the join graph of a query Q and the edges of its query graph.

Lemma 4.1: Let JG_Q and QG be the join graph and the query graph of a query Q respectively. Then

(a) There is a one-to-one correspondence between the edges of JG_Q and the edges of QG.

(b) There is a one-to-one correspondence between the join clauses of Q and the edges of QG.

Proof: (a) Since any two relations R_i, R_j have at most one attribute in common, no two edges of JG_Q can map to the same edge in the query graph QG.

(b) By definition, there is a one-to-one correspondence between the join clauses of a query Q and the edges of its

join graph JG_Q. Then the result follows from (a). #

Since we consider only tree queries in this chapter, the following notations, unless stated otherwise, will be used throughout the chapter. Q is a tree query. JG_Q^T is the transitive closure of the join graph JG_Q of Q. Q^* is the query represented by JG_Q^T and QG^* is the query graph of Q^*. The following lemma states that a query is a tree query iff any reduced join graph for the query (i.e. any spanning forest of the transitive closure of its join graph) has a tree query graph.

Lemma 4.2: Q is a tree query iff the query graph of a reduced join graph is a tree.

Proof: If Q is a tree query then there is a spanning forest of JG_Q^T (i.e. a reduced join graph) having a tree query graph, by definition. If a reduced join graph for Q has a tree query graph then all the reduced join graphs must have tree query graphs since the query graphs corresponding to spanning forests of JG_Q^T have the same number of edges (by Lemma 4.1-(a)) and are connected (by Lemma 2.2). #

Any path in QG^* between two relations having an attribute a in common, contains only relations that have attribute a if no relation appears more than once in the path. Furthermore, any spanning tree of QG^* is a tree query graph of a query equivalent to Q. These are shown by the following lemma.

Lemma 4.3: For a tree query Q,

(a) If $P(R_i,R_j)$ is a path between R_i and R_j in QG^* such that no relation appears more than once in the path and R_i and R_j have attribute a in common, then every relation in the path $P(R_i,R_j)$ has attribute a.

(b) Any spanning tree of QG^* is a tree query graph of an equivalent query.

Proof: (a) Let $P(R_i,R_j)$ be a path in QG^*, where R_i and R_j have attribute a, but a relation R_t in the path does not have attribute a. By Lemma 2.1(a), the edges in $P(R_i,R_j)$ correspond to a unique set of edges, E, in JG_Q^T. Since no relation appears more than once in $P(R_i,R_j)$, there exists a spanning forest JG of JG_Q^T such that JG contains the edges in E and possibly some other edges. Let QG be the query graph of JG. We now show that there are at least two distinct paths between R_i and R_j in QG. Clearly, QG contains the path $P(R_i,R_j)$ which passes through R_t. Consider the unique path between $R_i.a$ and $R_j.a$ in the connected component of JG corresponding to the attribute a. This path is mapped to a unique path between R_i and R_j in QG which does not pass through R_t since R_t does not have attribute a. Thus there are at least two paths between R_i and R_j in QG, which contradicts the fact that QG is a tree (by Lemma 4.2).

(b) Let T be a spanning tree of QG^*. The edges of T correspond to a unique set of edges, U, in JG_Q^T, by Lemma 4.1(a). It is sufficient to show that U is a spanning forest

in JG_Q^T since any spanning forest of JG_Q^T is a join graph of a query equivalent to Q. Consider any two relations R_i and R_j in a connected component corresponding to an attribute, say a, in JG_Q^T. Since T is a spanning tree, there is a unique path, P, between R_i and R_j in T. By part (a), each relation in the path P has attribute a. Thus, the path P in T between R_i and R_j corresponds to a path between $R_i.a$ and $R_j.a$ in the connected component representing attribute a in JG_Q^T. Since the above argument is true for every pair of relations in a connected component in JG_Q, U covers at least a spanning forest of JG_Q^T. #

Given a tree query Q with a target list, the attributes contained in the target list are called <u>output attributes</u> and the relations with the output attributes are called <u>output relations</u>. The site of the network at which the answer of the query is required is called the <u>result node</u>. A relation R is said to be <u>fully reduced</u> with respect to Q if all the tuples of R which do not satisfy the qualification of Q are eliminated. For a tree query Q, it has been shown [3] that any relation R can be fully reduced with respect to Q by a sequence of semi-joins traversing+ the tree query graph of an equivalent query from the leaves to the root R in a breadth-first order.

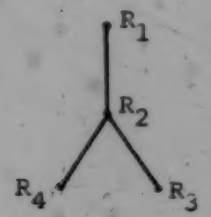
+A semi-join $R_i \dashrightarrow R_j$ traverses the edge (R_i, R_j) of a query graph from R_i to R_j.

Example 4.3: Consider the following query Q where

$Q = (R_1.a = R_2.a).\underline{and}.(R_2.a = R_3.a).\underline{and}.(R_2.b = R_4.b)$

The query graph QG of Q is

QG:



A sequence of semi-joins $z = \langle R_4 \rightarrow R_2,\ R_3 \rightarrow R_2,\ R_2 \rightarrow R_1 \rangle$ traverses QG from the leaves to the root R_1 in breadth-first order. After the first two semi-joins $R_4 \rightarrow R_2$, $R_3 \rightarrow R_2$ are executed, the tuples of R_2 which do not satisfy $Q_{sub} = (R_4.b = R_2.b).\underline{and}.(R_3.a = R_2.a)$ are eliminated. That is, R_2 is fully reduced with respect to Q_{sub} which is the subquery of Q corresponding to the subtree of QG with root R_2. Similarly, after the last semi-join $R_2 \rightarrow R_1$ is executed, R_1 is fully reduced with respect to Q since the tuples of R_2 satisfy the subquery before the semi-join $R_2 \rightarrow R_1$. #

4.2 Problem Formulation

The problem of optimizing tree queries by semi-joins is that of finding the optimum sequence of semi-joins fully reducing the output relations with respect to the given query. After the output relations are fully reduced, their projections over the output attributes can be transmitted to the result node.

The cost of a semi-join $R_i \rightarrow R_j$, denoted by $Cost(R_i \rightarrow R_j)$, is the cost of data transmission (from the site containing R_i to that of R_j) required to compute the semi-join, i.e.

$$Cost(R_i \rightarrow R_j) = C_0 + C_1 . w_a . |R_i[a]|,$$

where a is the attribute common to R_i and R_j, $|R_i[a]|$ is the number of distinct values in the column a of relation R_i, w_a is the average width of a data value in that column, C_0 and C_1 are fixed constants. The size of $R_i[a]$ is the product of the number of distinct values in the column of the relation R_i corresponding to the attribute a and the width of that column. The cost of a strategy, i.e. a sequence semi-joins, is the sum of the costs of the semi-joins employed in the strategy. (This cost criterion is called total time cost in [13]). It turns out that we need to estimate the number of distinct values in a column of a relation after one of its columns is reduced by a semi-join. The estimation can be done using the following assumptions.

Assumption 1: The values in a column of a relation are uniformly distributed; and the values in two different columns in different relations corresponding to a common attribute are independent. This is the same assumption used in [13].

Assumption 2: If the number of distinct values in one column of a relation is reduced, the number of distinct values in each of the other columns of the same relation is reduced proportionally. In [13], it is assumed that, after a

column is reduced, the number of distinct values in other columns of the same relation is unchanged. Both assumption 2 and the assumption in [13] may lead to some inaccuracies.

In this chapter, assumption 2 is used only to illustrate the optimization procedure (to be presented) which is also valid for other estimation methods. The estimation of the amount of data transfer required by a sequence of semi-joins using the assumptions 1 and 2 is illustrated in the following example.

Example 4.4: Consider the query Q of example 4.3, where $Q = (R_1.a = R_2.a).\underline{and}.(R_2.a = R_3.a).\underline{and}.(R_2.b = R_4.b)$ and the sequence $z_1 = \langle R_1 \dashrightarrow R_2,\ R_2 \dashrightarrow R_3 \rangle$. Let A and B be the domains corresponding to attributes a and b respectively. Let w_a and W_b be the width of a data value in the domains A and B respectively. Then,

$$Cost(z_1) = C_0 + C_1 \cdot w_a \cdot |R_1[a]| + C_0 + C_1 \cdot w_b \cdot |R_2'[a]|,$$

where $|R_2'[a]|$ is the expected cardinality of $R_2[a]$ after $R_1 \dashrightarrow R_2$ is executed. Let $|A|$ be the cardinality of domain A. By Assumption 1, the expected number of distinct values in $R_2[a]$ satisfying $R_1.a = R_2.a$ is $p_{a_1} * |R_2[a]|$ where $p_{a_1} = |R_1[a]|/|A|$ is the probability that a given value in domain A is in $R_1[a]$. Thus, $|R_2'[a]| = p_{a_1} \cdot |R_2[a]|$. Similarly, the estimated cardinality $|R_3'[a]|$ of $R_3[a]$ after z_1 is executed is $|R_3'[a]| = p_{a_1} \cdot p_{a_2} \cdot |R_3[a]|$ $= p_{a_1} \cdot p_{a_2} \cdot p_{a_3} \cdot |A|$. Consider a sequence $z_2 = \langle R_3 \dashrightarrow R_1,\ R_1 \dashrightarrow R_2,\ R_2 \dashrightarrow R_3 \rangle$. The estimated cardinality $|R_3'[a]|$ of $R_3[a]$ after

z_2 is executed is also given by $p_{a_1} \cdot p_{a_2} \cdot p_{a_3} \cdot |A|$, since R_3 can not reduce itself.

Consider the sequence $z_3 = \langle R_1 \rightarrow R_2, R_2 \rightarrow R_4 \rangle$.

$Cost(z_3) = 2C_0 + C_1(w_a \cdot |R_1[a]| + w_b \cdot |R_2'[b]|)$,

where $|R_2'[b]|$ is the estimated cardinality of $R_2[b]$ after the semi-join $R_1 \rightarrow R_2$ is executed. The cardinality of $R_2[a]$ after $R_1 \rightarrow R_2$ is estimated to be $p_{a_1} \cdot |R_2[a]|$ (by assumption 1) and then $R_2'[b]$ is $p_{a_1} \cdot |R_2[b]|$ (by assumption 2). Thus, after z_2 is executed, the number of distinct values in $R_4[b]$ becomes $p_{a_1} \cdot p_{b_2} \cdot |R_4[b]|$. #

In this chapter, we first consider tree queries with a single output relation, and find the optimum sequence of semi-joins fully reducing the output relation. In section 4.4.3, the optimum sequence of semi-joins fully reducing any one of the relations referenced by a tree query will be given for queries in which every relation has different output attributes.

Let R_r be the output relation specified by the query Q. If R_r resides at a result node then the optimum strategy fully reducing R_r at its site clearly answers Q with minimum cost. If the result node of the query does not contain any one of the relations referenced in Q then the optimum strategy answering Q consists of the optimum sequence of semi-joins fully reducing R_r at its site and then transmitting the projection of R_r over the output attributes

to the result node.

The above problem was studied by Hevner and Yao[13], but for a small class of tree queries called simple queries. In a simple query, each relation has exactly one attribute and that attribute is common to all the relations in the query.

Let S = {R_1,...,R?} be the set of relations referenced by Q. A transmission strategy, denoted T(R_i,S), is a sequence of semi-joins fully reducing relation R_i with respect to Q. A sequence of semi-joins $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$,..., $R_{k-1} \rightarrow R_k$ forms a transmission path $R_1 \rightarrow \ldots \rightarrow R_k$ from R_1 to R_k of length k. A relation R_j may occur in a transmission path more than once. In order to differentiate between different occurrences of the same relation, occurrences of relations in a transmission path are represented by distinct nodes. For example, $R_j \rightarrow R_k \rightarrow R_j$ is represented by $N_j \rightarrow N_k \rightarrow N_t$ where nodes N_j, N_k represent R_j and R_k, respectively, and N_t represents the second occurrence of R_j in the path. For notational convenience, N_j and R_j will be used interchangeably when there is no ambiguity. In a transmission path $N_j \rightarrow N_k$, N_j is the immediate predecessor of N_k and N_k is the immediate successor of N_j. If the transmission path $P(N_j,N_k)$ from N_j to N_k is of length ≥ 1 then N_j is a predecessor of N_k, and N_k is a successor of N_j. Clearly, if an occurrence of R_j is an immediate predecessor of an occurrence of R_k then R_j and R_k must have a common

attribute. However, one of the predecessors of a relation R_k in a transmission path may not have an attribute common to R_k. If a node N_j has more than one immediate predecessor, say $N_{j_1},\ldots,N_{j_t}$, $t>1$, all incoming semi-joins, i.e $N_{j_s}\rightarrow N_j$, $1\leq s\leq t$, are executed before any outgoing semi-joins of the form $N_j\rightarrow N_t$.

The following two lemmas prove some useful properties of transmission paths that will be utilized later.

Lemma 4.4: Let $N_1\rightarrow N_2\rightarrow N_3$ be a transmission path where N_i is an occurrence of R_i, $1\leq i\leq 3$, and R_1, R_3 have no attribute in common. Then any transmission path $P(N_1,N_3)$ contains an occurrence of R_2.

Proof: Let R_1, R_2 have attribute a in common and R_2, R_3 have attribute b in common. Suppose there is a path $P(N_1,N_3)$ which does not contain any occurrence of R_2. Let N_j be the last occurrence of a relation with attribute a (see Fig. 4.1) in the path $P(N_1,N_3)$, i.e. $N_j=N_1$ if there is no such occurrence other than N_1. Then $R_j\neq R_2$ since R_2 does not occur in $P(N_1,N_3)$. $R_j\neq R_3$ since R_3 does not have attribute a. Since N_j is the last occurrence of a relation with attribute a in $P(N_1,N_3)$, the path from N_2 to N_j via N_3 does not contain any intermediate relation having attribute a. This contradicts with Lemma 4.3(a). #

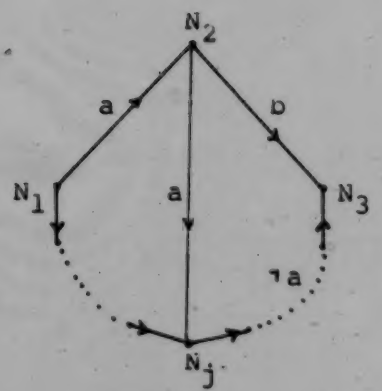


Figure 4.1. Illustration for Lemma 4.4

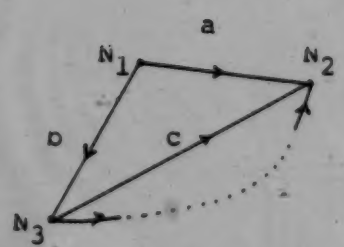


Figure 4.2. Illustration for Lemma 4.5.

Lemma 4.5: Let R_1 and R_2 be two relations having attribute a in common and $P(N_1,N_2)$ be any transmission path where N_1 is the only occurrence of R_1 in the path and N_2 is an occurrence of R_2. Then R_1 and its immediate successor N_3 in $P(N_1,N_2)$ also have attribute a in common.

Proof: Suppose N_3 does not have attribute a. Then there are two cases:

Case 1: N_3 and R_2 have no attribute in common. Since $N_3 \dashrightarrow N_1 \dashrightarrow N_2$ is a possible transmission path, the subpath of $P(N_1,N_2)$ from N_3 to N_2 must contain an occurrence of R_1 (by Lemma 4.4). This contradicts with N_1 being the only occurrence of R_1 in the path $P(N_1,N_2)$ via N_3.

Case 2: N_3 and R_2 have attribute c in common, $c \neq a$. Then there is a path between R_1 and R_2 via N_3 where N_3 does not have attribute a. This contradicts with Lemma 4.3(a). #

A transmission strategy $T(R_r,S)$ is represented by a directed graph whose vertices are the nodes representing occurrences of relations in the strategy and whose arcs represent the semi-joins. For notational convenience, we use $T(R_r,s)$ to denote both the transmission strategy and its graph representation. Let N_r be the last occurrence of R_r in $T(R_r,S)$, i.e. N_r is not a predecessor of any occurrence of R_r. It will be shown (Lemma 4.6) that there must be a transmission path in $T(R_r,S)$ from an occurrence of every relation R_j, $j \neq r$, in S to N_r. Moreover, the semi-joins in $T(R_r,S)$ traverse a tree query graph of an equivalent query

from the leaves to the root R_r such that the root of each subtree of QG is fully reduced with respect to the subquery corresponding to that subtree by a substrategy of $T(R_r,S)$ (Lemma 4.7). These are illustrated by the following example.

Example 4.5: Consider the query Q in example 4.4, where

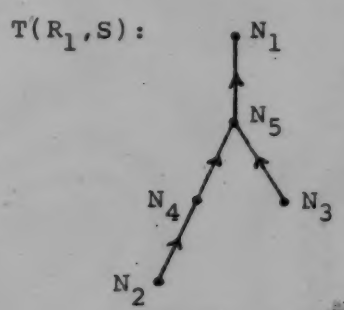
$$S = \{R_1(a),\ R_2(a,b),\ R_3(a),\ R_4(b)\}.$$

Consider two sequences of semi-joins below.

$z_1 = \langle R_2 \dashrightarrow R_4,\ R_3 \dashrightarrow R_2,\ R_4 \dashrightarrow R_2,\ R_2 \dashrightarrow R_1 \rangle$ and

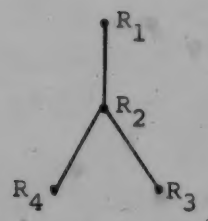
$z_2 = \langle R_2 \dashrightarrow R_4,\ R_3 \dashrightarrow R_2,\ R_2 \dashrightarrow R_1,\ R_4 \dashrightarrow R_2 \rangle$.

z_1 is a sequence of semi-joins fully reducing R_1 with respect to Q, i.e. $z_1 = T(R_1,S)$. The directed graph representing $T(R_1,S)$ is

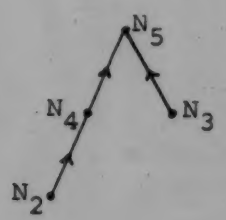


where N_5 represents the second occurrence of R_2 in the strategy. For each relation R_j, $2 \le j \le 4$, there is a transmission path from an occurrence of R_j to the last occurrence of R_1, i.e. N_1, in $T(R_1,S)$. The tree query graph QG traversed by $T(R_1,S)$ from the leaves to the root R_1 is as shown below.

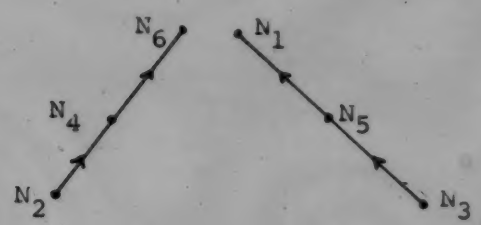
QG:



Note that the edge (R_2, R_4) of QG is traversed twice in $T(R_1,S)$. The subquery corresponding to the subtree of QG with root R_2 is $Q_{sub} = (R_4.b = R_2.b).\underline{and}.(R_3.a = R_2.a)$. The substrategy of $T(R_1,S)$ which fully reduces R_2 with respect to the subquery Q_{sub} is given below.



Consider z_2. The transmission paths in z_2 are shown below



where N_5 and N_6 represent the second and the third occurrences of R_2. Note that, there is no transmission path from an occurrence of R_4 to the last occurrence of R_1 in z_2. Thus, z_2 is <u>not</u> a transmission strategy fully reducing R_1 with respect to Q. #

As discussed before, if a sequence of N-1 semi-joins z

traverses a tree query graph QG of an equivalent query from the leaves to the root R_i then z fully reduces R_i with respect to Q [3]. Before the root R_i is fully reduced with respect to Q, the root of each subtree of QG is fully reduced by z with respect to the subquery corresponding to that subtree (see Example 4.3).

Lemma 4.6 below shows that <u>any</u> sequence of semi-joins fully reducing a relation R_i necessarily contains a <u>subsequence</u>[+] of z' with N-1 semi-joins such that z' traverses a tree query graph QG of an equivalent query from the leaves to the root R_i, and the root of each subtree of QG is fully reduced by z' with respect to the subquery corresponding to that subtree.

Let z be a sequence of semi-joins. Consider the following algorithm to obtain a subsequence z' of z such that z' fully reduces R_r with respect to Q and has N-1 semi-joins if z fully reduces R_r with respect to Q.

<u>Algorithm 4.1</u>:

<u>Input</u>: A sequence $z = \langle SJ_1, \ldots, SJ_m \rangle$ of semi-joins fully reducing R_r with respect to Q.

[+] A subsequence z' of a sequence of semi-joins z contains a subset of the semi-joins in z such that their relative order of appearance in z is preserved in z'. For example

$$z' = \langle R_3 \dashrightarrow R_2,\ R_4 \dashrightarrow R_2,\ R_2 \dashrightarrow R_1 \rangle$$

is a subsequence of z_1 in Example 4.5, however z_2 in Example 4.5 is not a subsequence of z_1.

Output: A subsequence z' of z such that z' fully reduces R_r and has exactly N-1 semi-joins.

(1) Let $S'=\{R_r\}$, z'=null sequence.

(2) Starting from the last semi-join in z for each semi-join $SJ_k = R_j \rightarrow R_t$ repeat

begin delete SJ_k from z;

If $R_j \notin S'$ and $R_t \in S'$ then

begin

$S' := S' \cup \{R_j\}$;

$z' := S_k | z'$;+

end;

end

until all the semi-joins in z are examined.

(3) Stop.

Since the number of semi-joins in z is finite, the algorithm terminates. The following lemma proves the correctness of the algorithm.

Lemma 4.6: Let $S = \{R_1,\ldots,R_N\}$ be the set of relations referenced by a tree query Q, z be a sequence of semi-joins involving relations in S and z' be the subsequence of z found by algorithm 4.1.

(a) If $z = \{SJ_1,\ldots,SJ_N\}$ is a transmission strategy $T(R_r,S)$ then z' has exactly N-1 semi-joins and fully reduces R_i with respect to Q. Moreover, for every

+ $S_k | z'$ denotes the concatenation of S_k with z'.

relation R_j, $j \neq r$, in S, z' contains a transmission path from R_j to R_r.

(b) If z contains a transmission path from every relation R_j, $j \neq r$, in S to R_r then z' also contains a transmission path from every relation R_j to R_r.

Proof: (a) Let z_k, z'_k, and S'_k denote z, z' and the set S' respectively before step (2) of algorithm 4.1 is executed for the semi-join SJ_k in z, $1 \leq k \leq m$. By backward induction on k we will show that at each stage k of the algorithm, $z_k | z'_k$ fully reduces R_r, z'_k has exactly $|S'_k|-1$ semi-joins and for each relation R_j in S'_k, $j \neq r$, z'_k has a transmission path from R_j to R_r.

Basis: k=m, trivially true.

Induction step: Suppose it is true before SJ_k in z is examined, $1 \leq k \leq m$. We will show that it is also true after SJ_k is examined. Let SJ_k in z be $R_j \dashrightarrow R_t$. There are two cases:

Case 1: $R_j \notin S'_k$ and $R_t \in S'_k$. Then $R_j \dashrightarrow R_t$ is deleted from z_k and concatenated to z'_k. That is after SJ_k is examined $z_{k-1} = \langle SJ_1, \ldots, SJ_{k-1} \rangle$ and $z'_{k-1} = SJ_k | z'_k$. Since $z_k | z'_k$ is the same as $z_{k-1} | z'_{k-1}$, by induction hypothesis, $z_{k-1} | z'_{k-1}$ also fully reduces R_r with respect to Q. $S'_{k-1} = S'_k \cup \{R_j\}$. By induction hypothesis, z'_{k-1} contains a transmission path from each relation $R_s \in S'_{k-1}$, $s \neq r$, to R_r. Since the number of semi-joins in z'_k and the size of S'_k are both increased by 1 after the semi-join SJ_k is examined, z'_{k-1}, z'_{k-1} contains $|S'_{k-1}|-1$ semi-joins (by induction hypothesis).

Case 2: $R_j \in S'_k$ or $R_t \notin S'_k$. Then $R_j \rightarrow R_t$ is removed from z_k, but z'_{k-1} and S'_{k-1} are the same as z'_k and S'_k respectively. Since after the semi-join SJ_k is examined only z_k is changed, it suffices to show that $z_{k-1}|z'_{k-1}$ fully reduces R_i with respect to Q. There are three subcases.

(i) $R_j \in S'_k$ and $R_t \in S'_k$. By induction hypothesis, there is a transmission path in $z'_k = z'_{k-1}$ from R_j to R_r. Similarly, there is a transmission path in z'_{k-1} from R_t to R_r. Hence, $P(R_j,R_t) = R_j \rightarrow \ldots \rightarrow R_r \rightarrow \ldots \rightarrow R_t$ is a possible transmission path in z'_{k-1}. Since $R_j \rightarrow R_t$ is a semi-join, R_j and R_t has an attribute in common, say a. Then $R_j[a]$ is transmitted from R_j to R_r in the subpath from R_j to R_r of the path $P(R_j,R_t)$ (by Lemma 4.5), without employing the semi-join SJ_k. Since transmitting $R_j[a]$ to R_r more than once does not cause any further reduction on R_r, if $z_k|z'_k$ fully reduces R_r with respect to Q, so does $z_{k-1}|z'_k$ after the semi-join $R_j \rightarrow R_t$ is removed from z_k.

(ii) $R_j \in S'_k$ and $R_t \notin S'_k$. R_t is not transmitted to R_r after the semi-join $R_j \rightarrow R_t$ in $z_k|z'_k$. Hence, the semi-join SJ_k is not used in $z_k|z'_k$ in fully reducing R_r. Therefore, $z_{k-1}|z'_k$ also fully reduces R_r with respect to Q.

(iii) $R_j \notin S'_k$ and $R_t \notin S'_k$. Then neither R_t nor R_j is transmitted to R_r in $z_k|z'_k$ after the semi-join $R_j \rightarrow R_t$ is performed. Since $z_k|z'_k$ fully reduces R_r by induction hypothesis, $z_{k-1}|z'_k$ also fully reduces R_r.

when the algorithm terminates, there are no semi-joins left in z, i.e. $z_\emptyset$ is a null sequence and $z_\emptyset | z'_\emptyset = z'_\emptyset = z'$. Hence z' fully reduces R_r with respect to Q. Since z' contains exactly $|S'_\emptyset|-1$ semi-joins and fully reduces R_r with respect to Q, $S'_\emptyset = S$. Thus z' contains N-1 semi-joins and z' contains a transmission path from each relation R_j in S, $j \neq i$, to R_r.

(b) The proof is straightforward using induction on the number of semi-joins examined similar to part(a). #

Lemma 4.7: Let Q be a query with N relations and z' be a sequence of N-1 semi-joins. If z' contains a transmission path from each relation R_j, $j \neq r$, to R_r then there exists a tree query graph QG of an equivalent query traversed by z' such that the root of each subtree of QG is fully reduced with respect to the subquery corresponding to that subtree by z'.

Proof: Since QG^* contains an edge between any two relations having a common attribute, if a semi-join $R_j \rightarrow R_k$ is in z' then the edge (R_j, R_k) is in QG^*. Since there is a transmission path from each relation $R_j \in S$, $j \neq r$, to R_r in z', there is a one-to-one correspondence between the semi-joins in z' and the edges of a spanning tree of QG^*. By Lemma 4.3(b), any spanning tree of QG^* is a tree query graph of an equivalent query. Hence there is a one-to-one correspondence between the semi-joins in z' and the edges of a tree query graph QG of an equivalent query. For each relation R_j, $j \neq r$, there is exactly one transmission path

from R_j to R_r in z' and the semi-joins are executed in the order of their appearence in the path. Thus z' traverses QG from the leaves to the root R_r such that the root of each subtree of QG is fully reduced with respect to the subquery corresponding to that subtree. #

The following proposition gives the necessary condition to be satisfied by a sequence of semi-joins fully reducing a relation with respect to Q.

Proposition 4.1: Let $T(R_r,S)$ be a transmission strategy fully reducing a relation R_r with respect to a tree query Q, where S is the set of relations referenced by Q. Then

(a) For every relation R_j in S, $j \neq r$, there is a transmission path from N_j to N_r in $T(R_r,S)$, where N_j is an occurrence of R_j and N_r is the last occurrence of R_r in $T(R_r,S)$.

(b) There is a tree query graph QG of an equivalent query such that $T(R_r,S)$ traverses QG from the leaves to the root R_i, and the root of each subtree of QG is fully reduced with respect to the subquery corresponding to that subtree by a substrategy of $T(R_r,S)$.

Proof: Directly follows from Lemmas 4.6 and 4.7. #

Let Q be a tree query and $S = \{R_1,\ldots,R_N\}$ be the set of relations referenced by Q. Consider the attributes of a relation $R_i \in S$ that are named in the qualification of Q.

Each such attribute of R_i must be common[+] to at least one other relation in S since the qualification of Q is a conjunction of join clauses. Thus, the set S is sufficient to represent Q^* since the join clause $(R_i.a=R_j.a)$ is in Q^* iff relations R_i and R_j are in S and have attribute a in common. Clearly, equivalent queries have the same set S of relations. Hence, with no loss of generality, we can use the term "fully reduced over the set S" to mean "fully reduced with respect to the query Q".

Two transmission strategies are equivalent if both strategies fully reduce the same relation over the same set, S, irrespective of the contents of the relations in S. Clearly, the optimum strategy fully reducing a relation over S is the one with minimum cost in the set of all equivalent strategies.

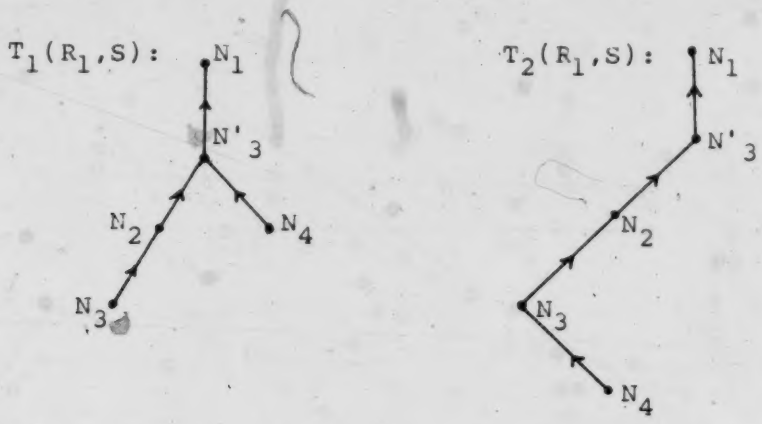
4.3 Potentially Optimum Strategies

A transmission strategy $T(R_r,S)$ fully reducing a relation over a set S is redundant if there is a lower cost equivalent strategy $T'(R_r,S)$, i.e. $Cost(T'(R_r,S)) \leq Cost(T(R_r,S)$ irrespective of the contents of the relations in S and there is at least one database state for relations in S for which $Cost(T'(R_r,S) < Cost(T(R_r,S))$. In this section, we indicate

[+]The common attributes have the same name, and any two relations have at most one attribute in common (see Section 4.1).

the configurations of potentially optimum strategies which are obtained by eliminating the redundant strategies from the set of all equivalent strategies.

Example 4.6: Let S = {$R_1(a)$, $R_2(a)$, $R_3(a,b)$, $R_4(b)$} be the relations referenced by a query Q. Consider the two equivalent strategies $T_1(R_1,S)$ and $T_2(R_1,S)$ below.



where N'_3 represents the second occurrence of R_3 in both strategies. $Cost(N_4 \rightarrow N'_3) = Cost(N_4 \rightarrow N_3)$. However, the cost of $N_3 \rightarrow N_2 \rightarrow N'_3$ is lower in $T_2(R_1,S)$ than in $T_1(R_1,S)$ since N_3 and N_2 are reduced by N_4 in $T_2(R_1,S)$ but not in $T_1(R_1,S)$. Consequently, $Cost(T_2(R_1,S)) \leq Cost(T_1(R_1,S))$ irrespective of the contents of the relations in S. Hence, $T_1(R_1,S)$ is a redundant strategy. *

Below a set of properties that must be satisfied by a potentially optimum strategy is presented. These properties allow us to discard redundant strategies and indicate what a potentially optimum strategy looks like.

Property 1: If $N_i \rightarrow N_j$ is a semi-join in $T(R_r,S)$ then N_i and N_j are occurrences of different relations.

Property 2: If $P_1(N_1,N)$ and $P_2(N_2,N)$ are two nonoverlapping[+] transmission paths in $T(R_r,S)$ intersecting at N then the relations represented by N_1 and N_2 do not have a common attribute.

Property 3: The outdegree of every node in $T(R_r,S)$ except the result node, N_r, representing the last occurrence of R_r is one and the outdegree of the result node is zero.

The first property is obvious since otherwise $T(R_r,S)$ contains a redundant semi-join, i.e. removing $N_i \rightarrow N_j$ results in a lower cost equivalent strategy. Property 2 implies that if any two relations R_i, R_j have an attribute in common then all occurrences of R_i and R_j must be in a single transmission path in $T(R_r,S)$. Let N_r be the result node of $T(R_r,S)$. There is a directed path in $T(R_r,S)$ from every node N_j to N_r (by prop. 4.1). Then, property 3 implies that $T(R_r,S)$ is an intree [12] with root N_r. Let $N_1,\ldots,N_t$, $t \geq 1$, be the immediate predecessors of N_r in $T(R_r,S)$ and S_i, $1 \leq i \leq t$, be the set of relations represented by the nodes in the subtree with root N_i (see figure 4.3). Each set S_i, $1 \leq i \leq t$, represents a subquery of Q^* where Q^* is the query represented by the set S. Let Q^*_i be the subquery of Q^*

[+]Two transmission paths are nonoverlapping if there is no arc contained in both paths.

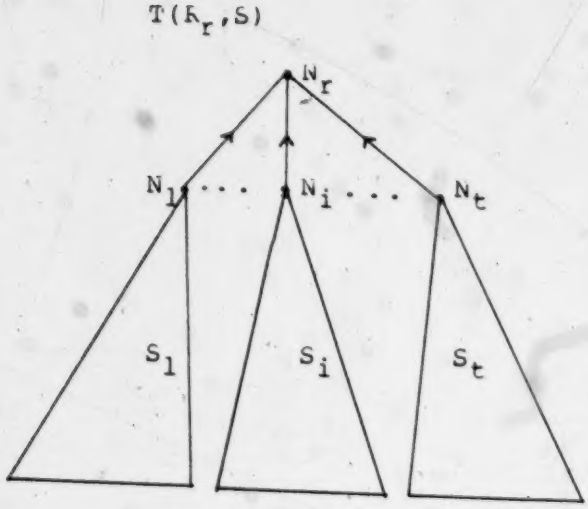


Figure 4.3. A strategy $T(R_r,S)$ and its substrategies, $T(R_i,S_i)$, $i=1,2,\ldots,t$.

represented by the set S_i, $1 \le i \le t$. Since Q^* is a tree query, Q^*_i is also a tree query. Since the subtree contains a transmission path from every relation in S_i to the last occurrence, N_i, of R_i, the subtree of $T(R_r,S)$ with root $R(i)$ is a substrategy fully reducing R_i with respect to Q^*_i (by Lemma 4.6(b) and Lemma 4.7). This is illustrated in Example 4.7 below.

Example 4.7: Consider the set $S = \{R_1(a), R_2(a), R_3(a,b), R_4(b)\}$ and the strategy $T(R_1,S)$ given in Example 4.6. The query Q^* represented by the set S is

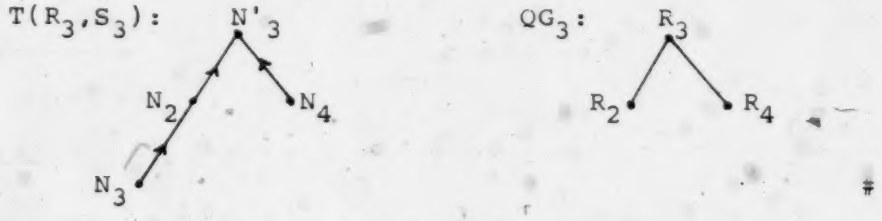
$$Q^* = (R_1.a = R_2.a).\underline{and}.(R_1.a = R_3.a).\underline{and}.(R_2.a = R_3.a) .\underline{and}.(R_3.b = R_4.b).$$

The subtree of $T_1(R_1,S)$ with root N'_3 is over the set $S_3 = \{R_2(a), R_3(a,b), R_4(b)\}$.

The subquery Q^*_3 of Q^* which is represented by S_3 is

$$Q^*_3 = (R_2.a = R_3.a).\underline{and}.(R_3.b = R_4.b)$$

The substrategy $T(R_3,S_3)$ and the tree query graph traversed by the substrategy are shown below.



#

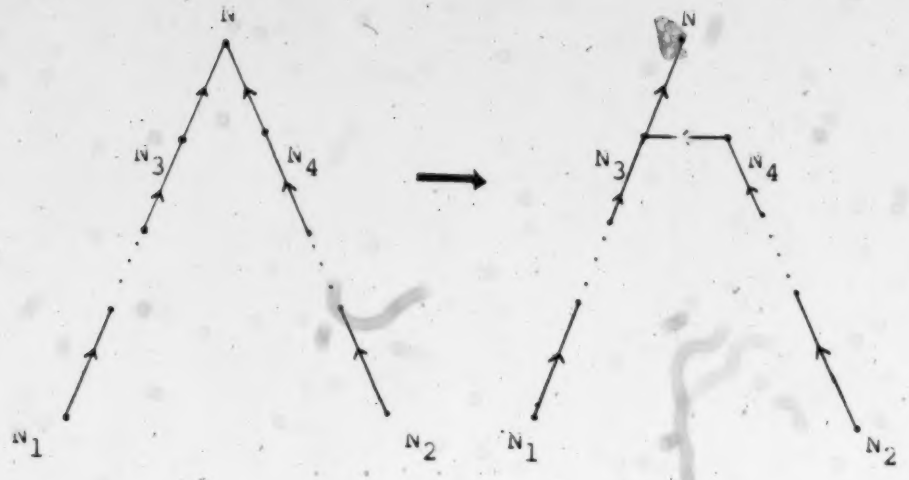
Properties 2 and 3 are proved by the following two lemmas.

Lemma 4.8: A potentially optimum strategy must satisfy property 2.

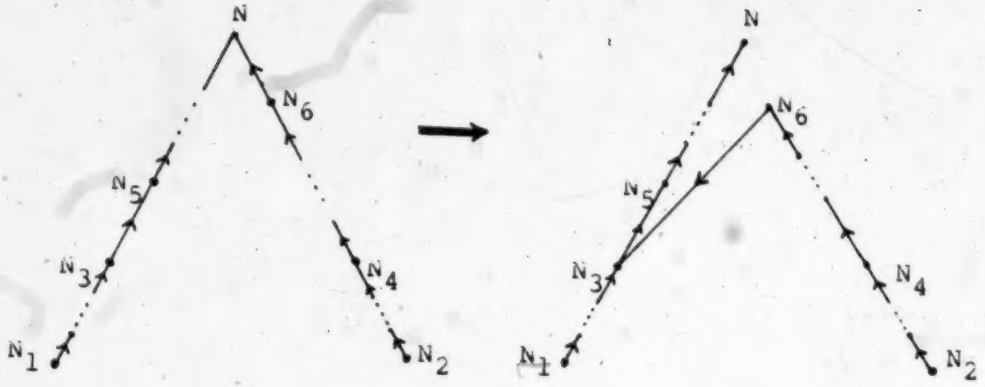
Proof: Let $P_1(N_1,N)$ and $P_2(N_2,N)$ be two nonoverlapping paths intersecting at N such that N_1, N_2 are occurrences of R_1, R_2 which have attribute a in common. Let N_3 and N_4 be the closest nodes to N in $P_1(N_1,N)$ and $P_2(N_2,N)$, respectively, that represent relations with attribute a. It suffices to show that there is a lower cost equivalent strategy having N_3 and N_4 as predecessors of N in a single path. There are two cases.

Case 1: N is the immediate successor of both N_3 and N_4 (see Fig. 4.4-(a)). Since N_3 and N_4 have attribute a in common, and N_4-->N-->N_3 is a possible transmission path, N_4 and N also have attribute a in common (by Lemma 4.5). Since any two relations have at most one common attribute, the data transmitted from N_4 to N by N_4-->N must be $N_4[a]$. Then replacing N_4-->N by N_4-->N_3 results in a lower cost equivalent strategy since $Cost(N_4\text{-->}N_3) = Cost(N_4\text{-->}N)$ while $Cost(N_3\text{-->}N)$ is lower in the new strategy because N_3 is reduced by the relations in the path from N_2 to N_4 as well.

Case 2: N is not the immediate successor of both N_3 and N_4 (see Fig. 4.4-(b)). Without loss of generality, let N_3 has N_5 as its immediate successor in $P(N_1,N)$ which represents R_5. Consider the path $N_3\text{-->}N_5\text{-->}\ldots\text{-->}N\text{-->}\ldots\text{-->}N_4$. Such a path is possible since $N_i\text{-->}N_j$ permits $N_j\text{-->}N_i$. If N_3 is the only occurrence of R_3 in the path then R_5 must have attribute a (by Lemma 4.5).



(a) Case 1



(b) Case 2

Figure 4.4. Illustration for Lemma 4.8.

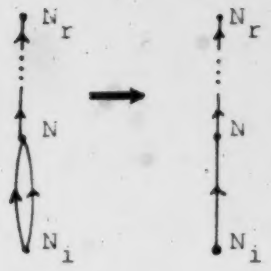
This will contradict with N_3 being the closest node to N having attribute a. Thus, R_3 occurs more than once in the path. Since N_3 and N_4 are the closest nodes to N in the path having attribute a, N must be an occurrence of R_3. Let N_6 be the immediate predecessor of N in $P_2(N_2,N)$. Since N_3 and N are the occurrences of the same relation, N_6-->N can be replaced by N_6-->N_3 resulting in an equivalent strategy in which N_3 and N_4 are predecessors of N in the same path. The new strategy has a lower cost since Cost(N_6-->N) = Cost(N_6-->N_3) while the cost of the path from N_3 to N in the new strategy is lower. #

Lemma 4.9: A potentially optimum strategy must satisfy property 3.

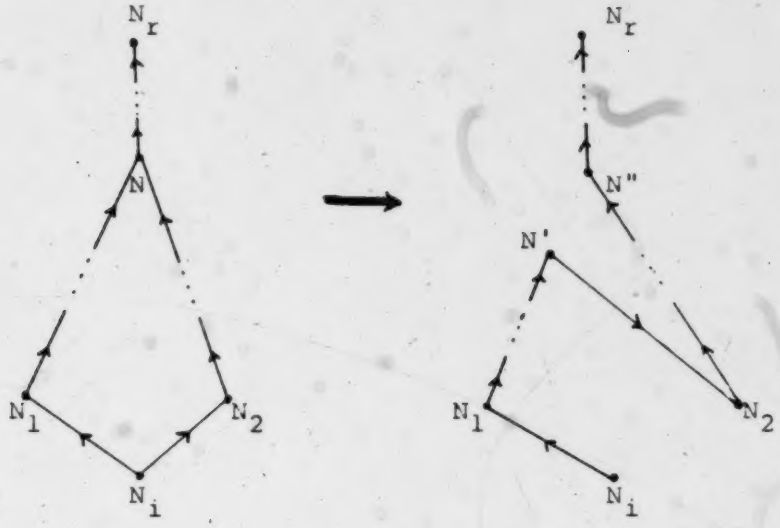
Proof: Let N_r be the result node of a strategy and N_i be any other node such that outdegree(N_i)=2. It is sufficient to show that this strategy can not be optimal. There are two transmission paths from N_i to N_r. Let N be the first node where the two paths $P_1(N_i,N_r)$ and $P_2(N_i,N_r)$ intersect (N may be N_r). There are two cases:

Case 1: The immediate successors of N_i in both paths are the same node, i.e. N (see Fig. 4.5-(a)). Then one of the semi-joins N_i-->N is redundant and can be removed, resulting with a lower cost equivalent strategy.

Case 2: The immediate successors of N_i are two different nodes, N_1 and N_2. By Property 2, N_1 and N_2 do not have any attribute in common. Since N_1-->N_i-->N_2 is a possible



(a) Case 1



(b) Case 2

Figure 4.5. Illustration for Lemma 4.9.

transmission path, $P(N_1,N_2) = N_1 \rightarrow \ldots \rightarrow N \rightarrow \ldots \rightarrow N_2$ must contain an occurrence of R_i (by Lemma 4.4). If the occurrence of R_i in $P(N_1,N_2)$ is between N_1 and N then property 2 is violated since R_i and N_2 have a common attribute. Similarly, the occurrence of R_i in $P(N_1,N_2)$ can not be between N and N_2. Thus, N must be an occurrence of R_i. Since N_i and N are the occurrences of the same relation, an equivalent strategy can be obtained by replacing N with two occurrences N' and N'' of R_i and replacing $N_i \rightarrow N_2$ and the path from N_2 to N by $N' \rightarrow N_2$ and the path from N_2 to N'' respectively (see Fig. 4.5-(b)). Since $Cost(N' \rightarrow N_2) \leq Cost(N_i \rightarrow N_2)$ and the nodes along the path from N_2 to N'' are reduced by additional relations, the resulting strategy has a lower cost. #

The following property states the conditions to be satisfied by substrategies of a potentially optimum strategy $T(R_r,S)$.

Property 4: Let $N_1,\ldots,N_t$, $t \geq 1$, be the immediate predecessors of N_r in $T(R_r,S)$, and S_i be the set of relations represented by the nodes of the subtree with root N_i, $1 \leq i \leq t$. Then

(a) $Indegree(N_r) \leq$ the number of attributes of R_r in S,

(b) If $indegree(N_r)=t$, $t>1$, then the sets $S_1,\ldots,S_t$ are disjoint subsets of $S-\{R_r\}$ such that $\bigcup_{i=1}^{t} S_i = S-\{R_r\}$,

(c) If $T(R_i,S_i)$ is a substrategy of $T(R_r,S)$ then $T(R_i,S_i)$ must be a potentially optimum strategy fully reducing

R_i over S_i.

Lemma 4.10: A potentially optimum strategy $T(R_r,S)$ must satisfy property 4.

Proof: (a) Suppose indegree(N_r) is greater than the number of attributes of R_r in S. Then there are at least two immediate predecessors N_i, N_j, $i \neq j$, of N_r in $T(R_r,S)$ such that N_i and N_j have a common attribute. This violates property 2 since N_i and N_j are in two nonoverlapping paths, $N_i \rightarrow N_r$ and $N_j \rightarrow N_r$.

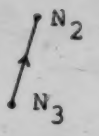
(b) If the sets $S_1,\ldots,S_t$, $t>1$, are not disjoint then there is a relation R_k contained in two or more sets, say S_i, S_j, $i \neq j$, $1 \leq i,j \leq t$. Let N_k and N'_k be the occurrences of R_k in S_i and S_j respectively. Since the path from N_k to N_r and the path from N'_k to N_r are two nonoverlapping paths, property 2 is violated. Similarly, $R_r \notin S_i$, $1 \leq i \leq t$, by property 2. Since R_r is fully reduced over S, $\bigcup_{i=1}^{t} S_i = S-\{R_r\}$ (by prop. 4.1).

(c) If a substrategy $T(R_i,S_i)$ of $T(R_r,S)$ is not potentially optimal then replacing $T(R_i,S_i)$ with a lower cost equivalent strategy results in a lower cost strategy equivalent to $T(R_r,S)$; a contradiction. #

At this point we want to differentiate a single attribute relation from a multiattribute relation because they satisfy different properties. A relation R_j is a multiattribute relation in S if R_j has more than one joining attribute in S; otherwise it is called a single attribute

relation in S. Let $T(R_i,S_i)$ be a substrategy of $T(R_r,S)$. It is possible that a multiattribute relation in S may become a single attribute relation in S_i as illustrated in the following example.

Example 4.8: Consider the strategy $T(R_3,S_3)$ in example 4.7, where $S_3 = \{R_2(a), R_3(a,b), R_4(b)\}$. The substrategy of $T(R_3,S_3)$ with result node N_2 is

$T(R_2,S_2)$:  where $S_2 = \{R_2(a), R_3(a)\}$.

R_3 has only one joining attribute, namely a, in S_2 since no other relation in S_2 has attribute b. That is, R_3 which is a multiattribute relation in S_3 becomes a single attribute relation in S_2. #

By property 4, indegree(N_r) is bounded by the number of attributes of R_r in S. If N_r is a multiattribute relation in S then N_r may have one or more immediate predecessors. Property 5, below, describes the substrategies of a potentially optimum strategy $T(R_r,S)$ when the result node N_r has more than one immediate predecessor.

Property 5: If R_r has t attributes in S and indegree(N_r)>1 in $T(R_r,S)$ then

(a) Indegree(N_r)=t

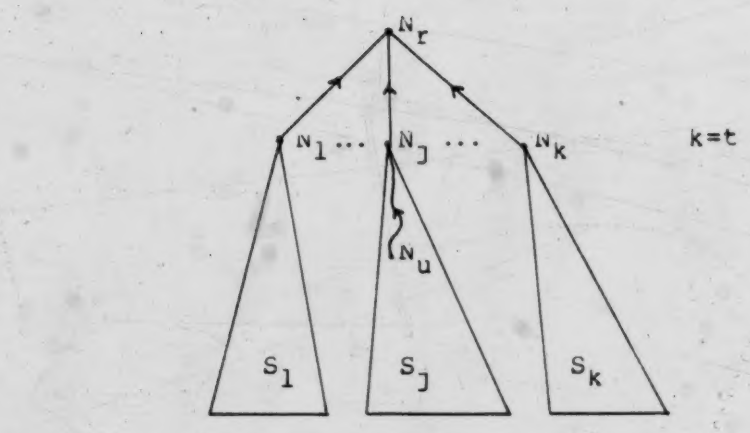
(b) The partition $S_1,\ldots,S_t$ of $S-\{R_r\}$ is unique, where S_i is the set corresponding to the substrategy of

$T(R_r,S)$ with result node N_i which is an immediate predecessor of N_r, $1 \leq i \leq t$.

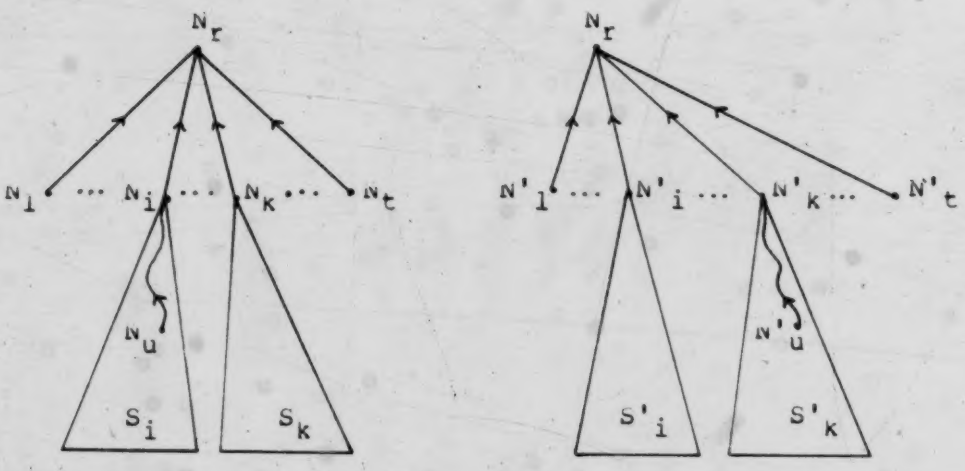
Lemma 4.11: A potentially optimum strategy $T(R_r,S)$ satisfies property 5.

Proof: (a) Let indegree(N_r) = $k>1$. Suppose $k<t$. Since any two relations can have at most one attribute in common, there is at least one attribute of R_r, say a, which is not common to any immediate predecessor N_i of N_r, $1 \leq i \leq k$. Since a is a joining attribute, there is at least one relation R_u, $u \neq r$, (see Figure 4.6) in a subtree with root N_j, $1 \leq j \leq k$, such that R_u has attribute a. Since $T(R_j,S_j)$ is a substrategy with result node N_j and $R_u \in S_j$, there is a transmission path $P(N_u,N_j)$ in $T(R_j,S_j)$ (by Prop. 4.1). If R_u and R_j have no common attribute then $P(N_u,N_j)$ must contain an occurrence of R_r (by Lemma 4.4) since $N_u \rightarrow N_r \rightarrow N_j$ is a possible transmission path. But this contradicts with $R_r \notin S_j$ (by property 4-(b)). Thus R_u and R_j must have an attribute in common. Since R_r and R_u have attribute a in common and $N_u \rightarrow N_j \rightarrow N_r$ is a possible transmission path, R_j must have attribute a (by Lemma 4.5). This is a contradiction since R_j does not have attribute a. Thus indegree(N_r) can not be less than t. Then indegree(N_r)=t since it can not be greater than t (by property 4-(a)).

(b) Let $\{N_1,\ldots,N_t\}$ and $\{N'_1,\ldots,N'_t\}$ be the sets of immediate predecessors of N_r in two different partitions $\{S_1,\ldots,S_t\}$ and $\{S'_1,\ldots,S'_c\}$ of $S-\{R_r\}$ respectively (see



(a) Case (a)



(b) Case (b)

Figure 4.6. Illustration for Lemma 4.11.

Fig. 4.6-(b)). By proof of part(a), c=t and each N'_i has a distinct attribute a_i in common with N_r, $1 \le i \le t$. By rearrangement of indices, if necessary, N_i and N'_i have attribute a_i in common, $1 \le i \le t$. By property 2, N_i and N_k $i \neq k$, $1 \le i,k \le t$, have no attribute in common. Since $N_i \rightarrow N_r \rightarrow N_k$ is a possible transmission path, any transmission path from R_i to R_k must contain an occurrence of R_r (by Lemma 4.4). we now show that this will be contradicted if $S_i \neq S'_i$ for some $1 \le i \le t$. The inequality of two sets implies that there is a relation $R_u \in S_i$ such that $R_u \notin S'_i$. Since

$\bigcup_{i=1}^{t} S_i = \bigcup_{i=1}^{t} S'_i = S-\{R_r\}$, there is a set S'_k such that $R_u \in S'_k$ for some $k \neq i$, $1 \le k \le t$. Let N_u and N'_u be the occurrences of R_u, and $P_1(N_u,N_i)$ and $P_2(N'_u,N'_k)$ be the two transmission paths in S_i and S'_k respectively. Then the path from N_i to N_u (the reverse of $P_1(N_u,N_i)$) joined with $N_u \rightarrow N'_u$, $P_2(N'_u,N'_k)$, $N'_k \rightarrow N_k$ forms a possible transmission path from R_i to R_k. This path does not contain R_r since neither S_i nor S'_k contains R_r; a contradiction. #

By property 4-(a), if R_r has t attributes in S, then indegree(N_r) $\le$ t. By property 5(a), if indegree(N_r) > 1 then it must be t. If indegree(N_r)=t in $T(R_r,S)$ then $S-\{R_r\}$ is partitioned into t disjoint subsets $S_1,\ldots,S_t$ each representing a subquery of Q^* and the partition is unique, by properties 4(b) and 5(b). The following property helps to

identify the immediate predecessor of N_r in $T(R_r,S)$ when R_r is a multiattribute relation in S but indegree(N_r)=1.

Property 6: Let $T(R_r,S)$ be a substrategy of $T(R_p,S_p)$ such that R_r is a multiattribute relation in S and R_p is the immediate successor of R_r. If indegree(N_r)=1 and R_i is the immediate predecessor of R_r then

(a) the substrategy $T(R_i,S_i)$ satisfies $S_i=S$,

(b) the attribute a_{pr} that is common to R_p and R_r is not the same as the attribute a_{ri} that is common to R_r and R_i.

Lemma 4.12: A potentially optimum strategy $T(R_j,S_j)$ satisfies property 6.

Proof: (a) Since R_r is a multiattribute relation in S, there is at least one attribute a_{rk}, $a_{rk} \neq a_{ri}$, of R_r such that a_{rk} is also an attribute of a relation R_k, $k \neq i$, in S and $T(R_r,S)$ contains an occurrence N_k of R_k. Since a_{ri} is an attribute of R_i, and R_r and R_i can have at most one attribute in common, R_i does not have a_{rk}. Thus $R_k \neq R_i$. Similarly, R_k does not have attribute a_{ri} of R_r because otherwise R_r and R_k would have two attributes in common. Then the occurrence N_k of R_k must be in the substrategy $T(R_i,S_i)$ with result node N_i since indegree(N_r)=1. Suppose R_i and R_k have an attribute, say b, in common. Since any two relations have at most one attribute in common, $b \neq a_{rk}$ and $b \neq a_{ri}$. Since $N_k \dashrightarrow N_r \dashrightarrow N_i$ is a possible transmission path, R_k and R_i must have attribute b in common (by lemma 4.5).

This is a contradiction since $b \neq a_{rk}$. Thus R_i and R_k have no attribute in common. Since $T(R_i,S_i$ is a strategy, there is a transmission path $P(N_k,N_i)$ in $T(R_i,S_i)$ (by prop. 4.1). By Lemma 4.4, $P(N_k,N_i)$ contains an occurrence of N'_r of R_r since $N_k \rightarrow N_r \rightarrow N_i$ is a possible transmission path. Therefore, S_i contains R_r, i.e. $S_i=S$.

(b) From part(a), there is at least one occurrence N'_r of R_r in $T(R_i,S)$. Consider the transmission path $P(N'_r,N_i)$ in the substrategy $T(R_i,S)$. with no loss of generality, let N'_r be the last occurrence of R_r in the path $P(N'_r,N_i)$. Then by Lemma 4.5, $R_r[a_{ri}]$ is transmitted to N_i via the immediate successor of N'_r in the path $P(N'_r,N_i)$. Suppose $a_{pr}=a_{ri}$. Consider the strategy obtained by replacing $N_i \rightarrow N_r \rightarrow N_p$ in $T(R_p,S_p)$ by $N_i \rightarrow N_p$. The new strategy (see Figure 4.7) is equivalent to $T(R_p,S_p)$ since $R_r[a_{ri}]$ transmitted to N_p by $N_r \rightarrow N_p$ is also transmitted to N_p via the transmission path from N'_r to N_p (i.e. $P(N'_r,N_i)$ joined with $N_i \rightarrow N_p$) in the new strategy. Since the cost of the new strategy is lower, this contradicts with the optimality of $T(R_p,S_p)$. #

Properties 1-6 are satisfied by all potentially optimum strategies irrespective of the size estimation method. We now give a property which uses assumption 1 (see Section 4.2). Consider two single attribute relations R_i, R_j in S, having the same joining attribute a. Let $|R_i[a]| < |R_j[a]|$. By Property 2, there is one transmission path in $T(R_r,S)$ containing all relations with attribute a. The following

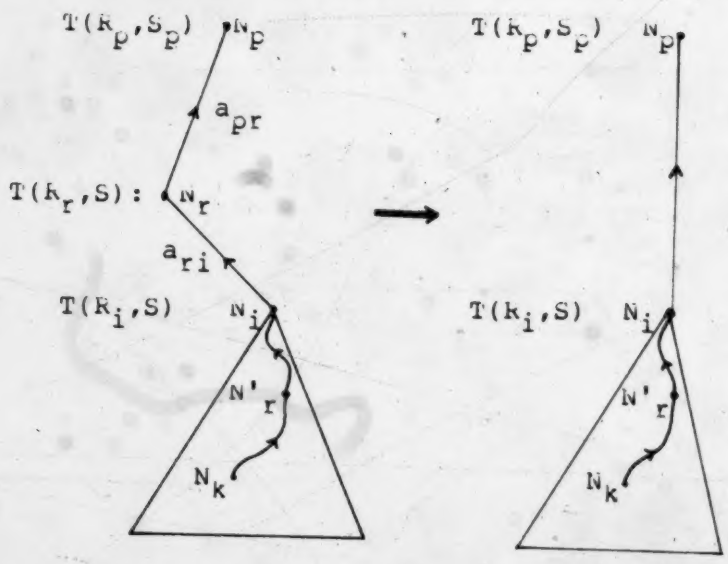


Figure 4.7. Illustration for Lemma 4.12.

property says that each single attribute relation, except possibly the result relation, occurs exactly once and in ascending order of size, i.e. R_i and R_j are in size order if R_i is a predecessor of R_j.

Property 7: Let $T(R_r,S)$ be a strategy fully reducing R_r over S, and a be an attribute of R_r. Then

(a) Every single attribute relation in S (except possibly R_r), occurs exactly once and in size order in $T(R_r,S)$.

(b) Let R_r be a single attribute relation. If R_r is the largest relation among single attribute relations having attribute a then the result node N_r of $T(R_r,S)$ is the only occurrence of R_r; otherwise, in addition to N_r, the substrategy of $T(R_r,S)$ may contain an occurrence of R_r in size order.

Lemma 4.13: A potentially optimum strategy $T(R_r,S)$ satisfies property 7.

Proof: (a) Let $N_1 \rightarrow \ldots \rightarrow N_i \rightarrow \ldots \rightarrow N_k \rightarrow N_j \rightarrow N_m \rightarrow \ldots \rightarrow N_r$ be a transmission path in $T(R_r,S)$. Suppose N_i, N_j are occurrences of a single attribute relation R_i with attribute a. The relations represented by N_k, N_m must have attribute a. Then replacing $N_k \rightarrow N_j \rightarrow N_m$ by $N_k \rightarrow N_m$ results in a lower cost equivalent strategy since $R_i[a]$ is transmitted to N_m in the new strategy with a lower cost. Thus, single attribute relations, except R_r, can not occur more than once in a transmission path. Since all occurrences of a relation are

in a single path (by property 2), there is exactly one occurrence of a single attribute relation (except R_r) in $T(R_r,S)$. Suppose N_i, N_j are occurrences of single attribute relations R_i, R_j with attribute a in common and $|R_i[a]| \geq |R_j[a]|$. Then interchanging N_i and N_j results in a lower cost equivalent strategy by assumption 1. This contradicts with the optimality of $T(R_r,S)$.

(b) Let R_r be a single attribute relation in S with attribute a. Then the arguments in (a) are also applicable to all occurrences of R_r in $T(R_r,S)$ except the result node N_r. Therefore, if R_r is not the largest single attribute relation with attribute a then $T(R_r,S)$ may contain another occurrence of R_r in size order in addition to N_r. #

The following example illustrates Property 7.

Example 4.9: Let Q be a simple query [13] with relations $S = \{R_1(a),\ldots,R_N(a)\}$. Suppose $|R_1[a]| \leq |R_2[a]| \leq \ldots \leq |R_N[a]|$. Let $T(R_r,S)$ be the optimum strategy fully reducing R_r over S. In [13], it is shown that, if $|R_r[a]|=|R_N[a]|$ then $T(R_r,S)$ is $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_r$. Otherwise,

either $R_1 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_N \rightarrow R_r$ (4.1)

or $R_1 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_r \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_N \rightarrow R_r$ (4.2)

is the optimum strategy $T(R_r,S)$.

If $|R_r[a]|=|R_N[a]|$ then $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_r$ is the only potentially optimum strategy satisfying properties 1-7,

hence it is the optimum strategy. Similarly, if $|R_r[a]|<|R_N[a]|$ then only (4.1) and (4.2) satisfy the properties 1-7 and, hence, one of them is the optimum. #

For a simple query Q with N relations, there are at most two potentially optimum strategies fully reducing a relation with respect to Q. However, for slightly more general queries, the number of potentially optimum strategies is exponential in spite of the restrictions imposed by properties 1-7, as illustrated below.

Example 4.10: Let Q be a query with a set of relations $S = \{R_1(a),\ldots,R_m(a),R_{m+1}(b),\ldots,R_{m+n}(b),R_{m+n+1}(a,b)\}$. Suppose the result relation is R_r. Since R_{m+n+1} is the only relation with two attributes, in $T(R_r,S)$, only R_{m+n+1} can have more than one immediate predecessor (by property 4(a)). Let NI_1 and NI_2 be the number of potentially optimum strategies $T(R_r,S)$ satisfying

(i) for any occurrence N_i of R_{m+n+1} in the strategy indegree$(N_i) \leq 1$,

(ii) there is at least one occurrence N_i of R_{m+n+1} in the strategy such that indegree(N_i),

respectively. Clearly the total number of potentially optimum strategies is NI_1+NI_2. We now show that

$$NI_1 \geq MIN\{\binom{m+n-1}{n},\binom{m+n-1}{m}\}.$$

In order to find NI_1, it is sufficient to observe that $T(R_r,S)$ consists of a single path and each single attribute relation (except possibly the result relation) occurs

exactly once, by property 7. Such a transmission strategy can be visualized as having m+n consecutive positions, where m of the positions are for $\{R_1,\ldots,R_m\}$ and n of the positions are for $\{R_{m+1},\ldots,R_{m+n}\}$; whenever two consecutive positions contain two single attribute relations having different attributes, R_{m+n+1} is assumed to be implicitly placed in between. Suppose the positions of $\{R_1,\ldots,R_m\}$ are fixed in the m+n positions. Then there is no choice in placing $\{R_{m+1},\ldots,R_{m+n+1}\}$ since they occur in ascending order of size. Thus, NI_1 is at least equal to the number of ways of choosing m positions for $R_1,\ldots,R_m$ out of m+n positions, i.e. $\binom{m+n}{m}$. If $R_r = R_m$ then the position of R_m is fixed and R_m occurs exactly once, hence there are at least

$$\binom{m+n-1}{m-1} = \binom{m+n-1}{n}$$

ways. Similarly, if $R_r = R_{m+n}$ then there are at least $\binom{m+n-1}{m}$ ways to place $R_1,\ldots,R_m$ into m+n-1 positions. #

In the next section, a dynamic programming approach will be presented to find the optimum strategy among the potentially optimum strategies fully reducing a relation with respect to a query Q.

4.4 Optimization

Consider a potentially optimum strategy $T(R_r,S)$ fully reducing a relation R_r with respect to Q. For each potentially optimum substrategy $T(R_j,S_j)$ of $T(R_r,S)$, the result relation R_j and the set S_j can be determined by properties 1-7, where $R_i \in S_i$ and $S_i \subset S$. The cost of $T(R_r,S)$ can be expressed in terms of the costs of its substrategies $T(R_j,S_j)$ and the costs of the semi-joins of the form $R_j \rightarrow R_r$. The costs of the substrategies of $T(R_i,S_i)$ can be determined recursively in terms of the costs of their substrategies, until substrategies of the form $T(R_j,S_j)$ with $S_j=\{R_j\}$ are obtained. Such a strategy consists of a single node, hence its cost is zero. This section first finds the optimum strategy fully reducing a specific relation R_r with respect to Q recursively, as outlined above. Then the optimum strategy fully reducing one of the relations with respect to Q is found. This, in turn, is utilized to find the optimum strategy fully reducing all the relations in Q.

4.4.1 Parameters Related to the Optimum Strategy

The optimum strategy fully reducing a relation R_r with respect to a query Q' (which may be a subquery of the given query Q or the query Q itself) is characterized by three parameters:

R_r: the result relation

S : the set of relations in Q'

a_{sr}: the attribute common to R_r and its immediate successor R_s. If Q' is the given query Q and R_r is the result relation specified by Q, then $a_{sr} = \emptyset$ since R_r does not have an immediate predecessor in this case.

Such an optimum strategy is denoted by $t(R_r,S,a_{sr})$. The parameter a_{sr} is utilized to ensure that the attribute common to R_r and its immediate predecessor R_i is different from that between R_i and its immediate predecessor (property 6(b)) when R_i is a multiattribute relation and indegree of R_i is 1.

4.4.2 Optimum Strategy Fully Reducing a Specified Relation

Let R_r be the result relation specified by the query, S be the set of relations in Q, and $t(R_r,S,a_{sr})$ be the optimum strategy fully reducing R_r over S. If R_r is a single attribute relation in S then there are two cases: (case 1) R_r is the largest of all single attribute relations in S having the same attribute as R_r and (case 2) R_r is not the largest of such relations in S. If R_r is a multiattribute relation in S with t attributes, t>1, then there are also two cases for R_r: (case 3) indegree(R_r)=t and (case 4) indegree(R_r)=1, by properties 4(a) and 5(a). Hence there are four cases for the result relation R_r in $t(R_r,S,a_{sr})$.

Consider a substrategy $t(R_j,S_j,a_{rj})$ of $t(R_r,S,a_{sr})$ where R_j is an immediate predecessor of R_r. By property 7,

all single attribute relations (except possibly R_r) appear exactly once and in size order in $t(R_r,S,a_{sr})$. Thus, If R_j is a single attribute relation in S then it must be the largest single attribute relation in S_j with that attribute (i.e. case 1 applies to R_j in $t(R_j,S_j,a_{rj})$). As illustrated in Example 4.8, a multiattribute relation in S may become a single attribute relation in S_j. Suppose the result relation R_j of $t(R_j,S_j,a_{rj})$ is a multiattribute relation in S but it becomes a single attribute relation in S_j. Let the attribute common to R_j and its immediate successor R_r be a. Since R_j becomes a single attribute relation in S_j, R_j does not have attribute a in S_j and there is no relation in S_j having attribute a. Suppose R_j has attribute b in S_j. If R_j is not the largest single attribute relation having attribute b in S_j then one can not replace R_j with a larger single attribute relation with attribute b in S_j since such a relation does not have attribute a and therefore can not be the immediate predecessor of R_r. Hence if R_j is a multiattribute relation in S but becomes a single attribute relation in S_j then R_j may or may not be the largest single attribute relation with that attribute in S_j (i.e. either case 1 or case 2 applies). If R_j is a multiattribute relation in S_j then either case 3 or case 4 applies to R_j in $t(R_j,S_j,a_{rj})$.

The optimal strategy $t(R_r,S,a_{sr})$ is described below in each of the four cases for R_r. For notational convenience,

the largest of all single attribute relations having the same attribute a in a set S is denoted by max(a,S).

Case 1: R_r is a single attribute relation with an attribute, say a_j, in S and $R_r = \max(a_j,S)$ (see Fig. 4.8).

By property 4-a, indegree(R_r)=1 in $t(R_r,S,a_{sr})$. Let R_j be the immediate predecessor of R_r and $t(R_j,S',a_{rj})$, $a_{rj}\neq a_j$, be the substrategy of $t(R_r,S,a_{sr})$. Since $R_r = \max(a_j,S)$, R_r does not occur in any substrategy of $t(R_r,S,a_{sr})$ (by property 7(b)). Thus, $S'=S-\{R_r\}$. Let $P_s(R_r)$ be the set of all possible immediate predecessors of R_r, i.e. $R_j \epsilon P_s(R_r)$. By property 7, R_j is either the largest single attribute relation having attribute a_j in S' or it is a multiattribute relation with attribute a_j in S'. Thus,

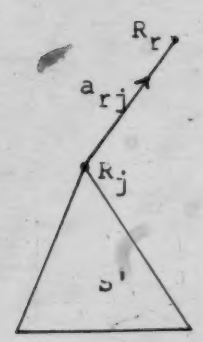
$$P_s(R_r) = \{\max(a_j,S-\{R_r\})\} \cup \{\text{multiattribute relations having attribute } a_j \text{ in } S\}.$$

The optimal strategy $t(R_r,S,a_{sr})$ is illustrated in figure 4.8.

The cost of $t(R_r,S,a_{sr})$, in terms of its substrategy is

$$\text{Cost } t(R_r,S,a_{sr}) = \min_{R_j \epsilon P_s(R_r)} \{\text{Cost } t(R_j,S',a_{rj}) + \bar{a}_{rj}(S')\} \qquad (4.3)$$

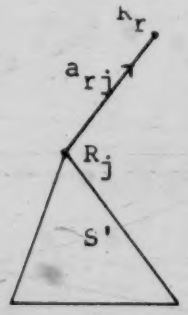
where $S'=S-\{R_r\}$, $\bar{a}_{rj}(S')$ is the cost of transmitting $R_j[a_{rj}]$ from R_j to R_r after it is fully reduced over S' by



$t(R_r,S,a_{sr})$

$t(R_j,S',a_{rj})$ where $S'=S-\{R_r\}$

Figure 4.8. The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 1.



$t(R_r,S,a_{sr})$

$t(R_j,S',a_{rj})$

wnere $S'=S$ or $S'=S-\{R_r\}$

Figure 4.9. The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 2.

$t(R_r,S,a_{sr})$

$t(R_j,S',a_{rj})$, $S'=S$

$t(R_r,S'',a_p)$, $S''=S$

Figure 4.10. Substrategies of $t(R_r,S,a_{sr})$.

$t(R_j,S',a_{rj})$. The value of $\bar{a}_{rj}(S')$ can be estimated using assumptions 1 and 2 as discussed in section 3.

Clearly, the substrategy $t(R_j,S',a_{rj})$ of $t(R_r,S,a_{rs})$ cannot contain $t(R_r,S,a_{rs})$ as its substrategy since S' does not contain R_r.

Case 2: R_r is a single attribute relation with an attribute, say a_j, and $R_r \neq \max(a_j,S)$ (see Fig. 4.9).

This case arises when R_r is the result node specified by the given query Q or R_r is a multiattribute relation in a higher level strategy, say $t(R_s,S_s,a_{ps})$, but it becomes a single attribute relation in S, where $t(R_r,S,a_{sr})$ is a substrategy of $t(R_s,S_s,a_{ps})$.

By property 4-(a), indegree(R_r)=1. The result node R_j of the substrategy $t(R_j,S',a_{rj})$ of $t(R_r,S,a_{sr})$ must be a member of $P_s(R_r)$ which is the same as defined in Case 1. However, S' is either S or S-$\{R_r\}$ (by property 8).

The cost of $t(R_r,S,a_{sr})$, in terms of its substrategy is

$$\text{Cost } t(R_r,S,a_{sr}) = \min_{S' \in \{S,\, S-\{R_r\}\}} \{ \min_{R_j \in P_s(R_r)} \{\text{cost } t(R_j,S',a_{rj}) + \bar{a}_{rj}(S')\}\} \quad (4.4)$$

where the parameters are as defined before.

Clearly, if S'=S-$\{R_r\}$, the substrategy $t(R_j,S',a_{rj})$

does not contain $t(R_r,S,a_{sr})$ as its substrategy. We now show that if S'=S then $t(R_j,S',a_{rj})$ also does not contain $t(R_r,S,a_p)$ for any attribute a_p. Suppose S'=S and there is a substrategy $t(R_r,S'',a_p)$, S"=S, of $t(R_j,S',a_{rj})$. Consider the transmission path between the two occurrences of R_r (see Figure 4.10). Since $R_r \neq max(a_j,S)$, there is at least one single attribute relation R_k in S where $R_k=max(a_j,S)$. By property 7, R_k must be in the transmission path between the two occurrences of R_r since every single attribute relation in S appears exactly once and in size order except the occurrence of R_r which is the result node of $t(R_r,S,a_{sr})$. This implies that S" is a proper subset of S since there is no other occurrence of R_k in the substrategy with the result node R_k (by property 7). This contradicts with S"=S. Thus, $t(R_j,S',a_{rj})$ can not contain $t(R_r,S,a_p)$ as its substrategy.

Case 3: R_r is a multiattribute relation in S and indegree(R_r)>1 (see Fig. 4.11).

By Property 5, indegree(R_r) = number of attributes t of R_r in S, and there is a unique partition $S_1,\ldots,S_t$ of S-{R_r}, where S_j is the set of relations for a substrategy of $t(R_r,S,a_{sr})$, $1\leq j\leq t$, t>1. Let $X=\{R_1,\ldots,R_t\}$ be a set of immediate predecessors of R_r in $t(R_r,S,a_{sr})$. Each relation $R_j \in X$ has an attribute common to R_r, and (by property 2) no two relations in X have a common attribute. For notational convenience, let $R_j \in X$ be the result node of the substrategy over the set S_j, $1\leq j\leq t$. By property 7, R_j is

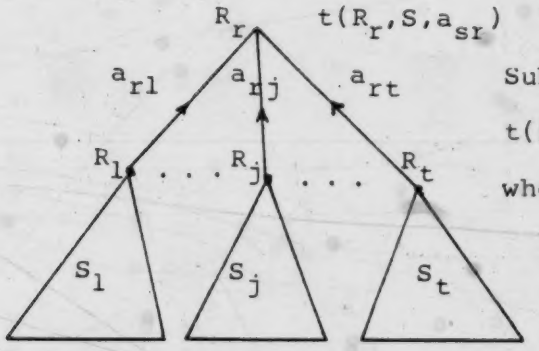


Figure 4.11. The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 3.

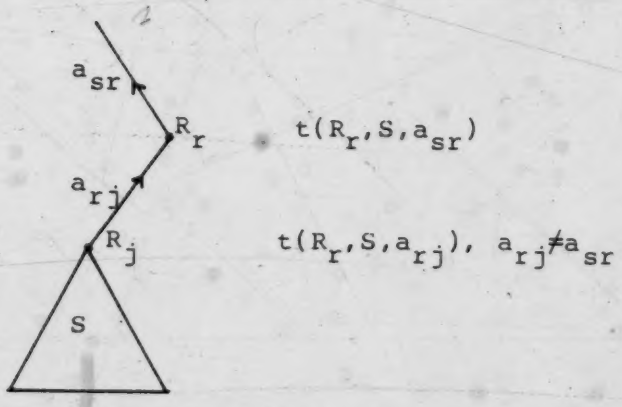


Figure 4.12. The Optimum Strategy $t(R_r,S,a_{sr})$ for Case 4.

either $\max(a_{rj},S_j)$ or a multiattribute relation with attribute a_{rj} in S_j. Thus given the set S and the set of attributes of R_r in S, the valid set of immediate predecessors X of R_r in $t(R_r,S,a_{sr})$ can be determined. Let $PX_s(R_r)$ be the set of all valid sets of immediate predecessors of R_r in S. Then the cost of $t(R_r,S,a_{sr})$ in terms of its substrategies is

$$\text{Cost } t(R_r,S,a_{sr}) = \min_{X \in PX_s(R_r)} \{ \sum_{R_j \in X} (\text{Cost } t(R_j,S_j,a_{rj}) + \bar{a}_{rj}(S_j)) \}. \quad (4.5)$$

Clearly, no substrategy $t(R_j,S_j,a_{rj})$ can contain $t(S,R_r,a_{sr})$ as its substrategy since $S_j \subset S-\{R_r\}$, $1 \le j \le t$.

Case 4: R_r is a multiattribute relation in S and indegree$(R_r)=1$ (see Fig. 4.12)

By property 6(a), the substrategy of $t(S,R_r,a_{sr})$ is also over the set S. Let R_j be the immediate predecessor of R_r in $t(S,R_r,a_{sr})$. Since R_r has t attributes $a_{r1},\ldots a_{rt}$ in S, $t>1$, R_j must be one of the relations in the set

$$P_s(R_r) = \bigcup_{j=1}^{t} \{\{\max(a_{rj},S)\} \cup \{\text{multiattribute relations in S having attribute } a_{rj}, \text{ except } R_r\}\}.$$

If R_r has an immediate successor, i.e. $a_{sr} \neq \emptyset$, then the immediate predecessor R_j of R_r is a relation in $P_s(R_r)$ such that R_j does not have attribute a_{sr} (by property 6(b)). Thus, the cost of $t(R_r,S,a_{sr})$ in terms of its substrategies

is

$$\text{Cost } t(R_r,S,a_{sr}) = \min_{\substack{R_j \in P_s(R_r) \\ a_{rj} \neq a_{sr}}} \{\text{Cost } t(R_j,S,a_{rj}) + \bar{a}_{rj}(S)\} \qquad (4.6)$$

We now show that $t(R_j,S,a_{rj})$ does not contain $t(S,R_r,a_{sr})$ as a substrategy, for any attribute a_{sr}. It suffices to consider the following different cases for the immediate predecessor R_j of R_r.

(i) R_j is a single attribute relation in S. Then $t(R_j,S,a_{rj})$ does not contain a substrategy over S with result node R_r (from Case (1) and Case (2)). Therefore, $t(R_j,S,a_{rj})$ can not contain $t(R_r,S,a_{sr})$.

(ii) R_j is a multiattribute relation in S and indegree(R_j)>1. Then $t(R_j,S,a_{rj})$ does not contain $t(R_r,S,a_{sr})$ since it does not contain any substrategy over S (from Case 3).

(iii) R_j is a multiattribute relation in S, and indegree(R_j)=1. Then applying the above arguments to the immediate predecessor of R_j recursively, the only case when $t(R_j,S,a_{rj})$ contains a substrategy over S which may contain $t(R_r,S,a_{sr})$ is when the immediate predecessor is a multiattribute relation in S with indegree one. Let R_j, $R_1,\ldots,R_t$ be such predecessors of R_j, where R_k, $1 \leq k \leq t$, is a multiattribute relation in S, indegree(R_k)=1 and the substrategy with the result relation R_k is over the set S (see figure

4.13). The following lemma shows that each multi-attribute relation in S can occur at most once in such a path $R_r \leftarrow\!- R_j \leftarrow\!- R_1 \leftarrow\!- \ldots \leftarrow\!- R_t$. Therefore, $t(R_j,S,a_{rj})$ can not contain $t(R_r,S,a_{sr})$ as its substrategy, i.e. equation (4.6) also converges.

Lemma 4.14: Let R_r, R_j, $R_1,\ldots,R_t$ be the multi-attribute relations in S and each is the result relation of a strategy over S and has one immediate predecessor such that $R_r \leftarrow\!- R_j \leftarrow\!- R_1 \leftarrow\!- \ldots \leftarrow\!- R_t$ is a transmission path in $t(R_r,S,a_{sr})$ (see Fig. 4.13). Then the relations $R_r,R_j,R_1,\ldots,R_t$ are different relations.

Proof: $R_r \neq R_j$ and $R_j \neq R_1$ (by property 1). Since $a_{rj} \neq a_{j1}$, $R_r \neq R_1$ because otherwise R_j and R_1 have two attributes in common. Suppose no relation occurs more than once in $R_r \leftarrow\!- R_j \leftarrow\!- R_1 \leftarrow\!- \ldots \leftarrow\!- R_k$, $1 \leq k \leq t-1$, but R_{k+1} is the same as one of the relations, say R_i, in the path $R_r \leftarrow\!- R_j \leftarrow\!- R_1 \ldots \leftarrow\!- R_i \leftarrow\!- \ldots \leftarrow\!- R_k \leftarrow\!- R_{k+1}$, where $i \in \{r,j,1,\ldots,k\}$. Let the immediate successor of R_k in the path be R_{k-1}, i.e. $k-1=j$ for $k=1$. Consider relations R_{k-1}, R_k and R_{k+1}. Since $a_{k-1,k} \neq a_{k,k+1}$, R_{k-1} and R_{k+1} can not have any common attribute (by Lemma 4.5). Then consider the path $R_i \leftarrow\!- R_{i+1} \leftarrow\!- \ldots \leftarrow\!- R_{k-1}$. Since $R_i = R_{k-1}$, one of the relations in the path must be R_k (by Lemma4.4). But this contradicts with the fact that $R_r,R_j,R_1,\ldots,R_k$ are all different relations. #

In each of the four cases, the cost of $t(R_r,S,a_{sr})$ is

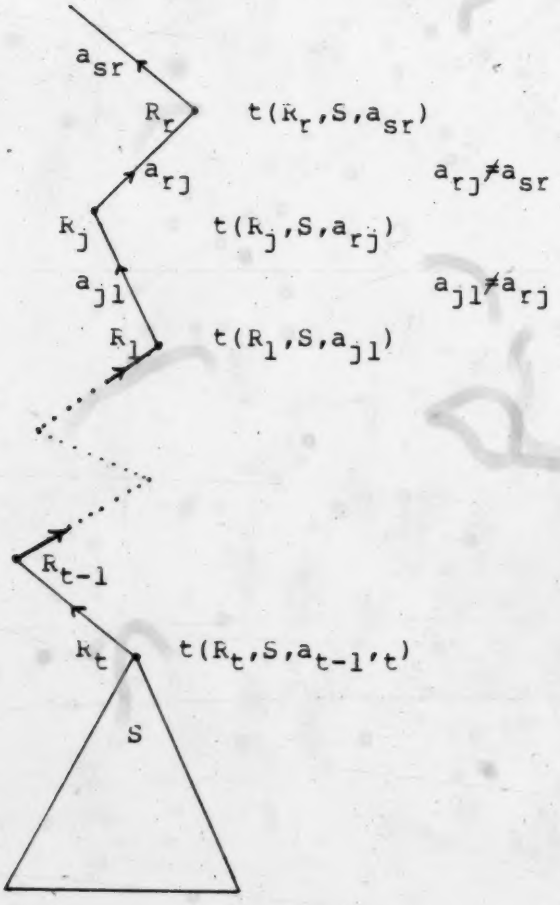


Figure 4.13. Illustration for Lemma 4.14.

eventually expressed in terms of the costs of substrategies over strictly smaller sets of relations. Consequently, equations (4.3)-(4.6) can be used recursively until the cost of $t(R_r,S,a_{sr})$ is expressed in terms of some constants and the costs of substrategies of the form $t(R_j,S_j,a_{kj})$, where $S_j=\{R_j\}$. The boundary conditions are

$$\text{Cost } t(R_j,\{R_j\},a_{ij}) = \emptyset$$

for all the relations R_j in Q and the joining attributes a_{ij} in Q.

The complexity of computing the optimum strategy $T(R_r,S) = t(R_r,S,\emptyset)$ is illustrated for the query given in Example 4.10. Without loss of generality, let the relations be $|R_1(a)|\leq\ldots\leq|R_m(a)|$, $|R_{m+1}(b)|\leq\ldots\leq|R_{m+n}(b)|$ and the result node be $R_{m+n+1}(a,b)$. Let $S_{k,j,1}=\{R_1(a),\ldots,R_k(a), R_{m+1}(b),\ldots,R_{m+j}(b), R_{m+n+1}(a,b)\}$. $\emptyset\leq k\leq m$, $\emptyset\leq j\leq n$, $k+j\geq 1$. In $S_{k,j,\emptyset}$ $R_{m+n+1}(a,b)$ is absent. The optimal strategy is $t(R_{m+n+1},S_{m,n,1},\emptyset)$.

The recursive algorithm can be visualized as having m.n stages. At stage (k,j), $1\leq k\leq m$, $1\leq j\leq n$, the optimal strategies over $S_{k,j,1}$ with result nodes R_k, R_{m+j}, R_{m+n+1} are expressed in terms of the optimal strategies at stages (k-1,j) and (k,j-1) (see Figure 4.14). using equations (4.3)-(4.6). More precisely, we compute the costs of strategies $t(R_k,S_{k,j,1},a)$, $t(R_{m+j},S_{k,j,1},b)$, $t(R_{m+n+1},S_{k,j,1},a)$ and $t(R_{m+n+1},S_{k,j,1},b)$. For example, in $t(R_k,S_{k,j,1},a)$ since R_k is the largest single attribute relation with attribute a in

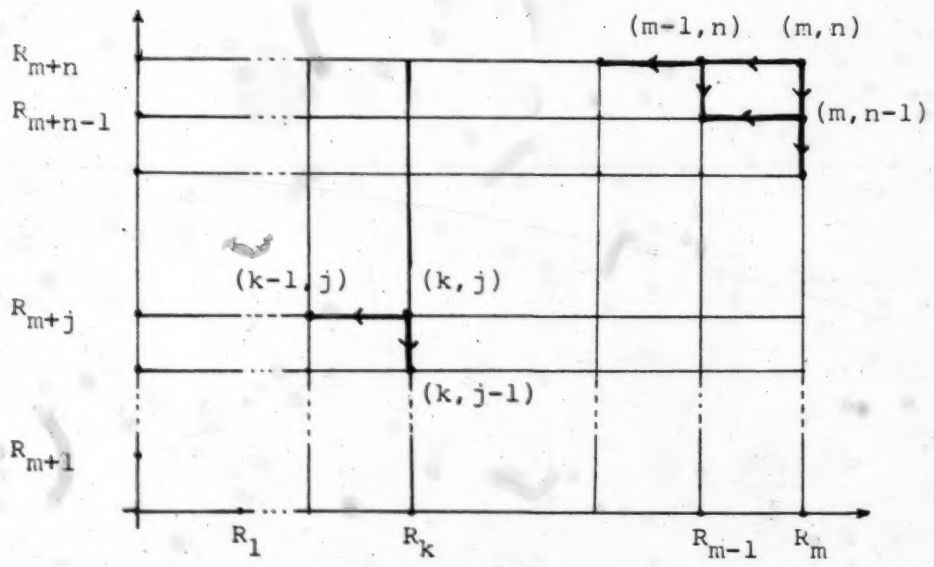


Figure 4.14. Illustration of the m.n Stages in Computing the Optimal Strategy over $S=S_{m,n,1}$.

$S_{k,j,1}$, $1 \leq k \leq m$, $1 \leq j \leq n$, its immediate predecessor is either R_{k-1} or R_{m+n+1}. Using eq. (4.3) we have

$$\text{Cost } t(R_k, S_{k,j,1}, a) = \text{MIN } \{\text{Cost } t(R_{k-1}, S_{k-1,j,1}, a), \text{Cost } t(R_{m+n+1}, S_{k-1,j,1}, a)\} + \bar{a}(S_{k-1,j,1}).$$

Since the data transferred to R_k is from a relation with attribute a and is reduced by $\{R_1, \ldots, R_{k-1}, R_{m+1}, \ldots, R_{m+j}, R_{m+n+1}\}$, using assumptions 1 and 2 for estimation as illustrated in Example 4.4, $\bar{a}(S_{k-1,j,1})$ is given by

$$C_\emptyset + C_1 . \{p_{a_1} \cdots p_{a_{k-1}} . p_{b_{m+1}} \cdots p_{b_{m+j}} . p_{a_{m+n+1}} . w_a . |A|\}$$

where p_{a_i} and $p_{b_{m+t}}$ are selectivities of column a of relation R_i and column b of relation R_{m+t} respectively, $1 \leq i \leq k-1$, $1 \leq t \leq j$.

The costs of $t(R_{m+j}, S_{k,j,1}, b)$ can be computed similarly. To compute the costs of $t(R_{m+n+1}, S_{k,j,1}, a)$ and $t(R_{m+n+1}, S_{k,j,1}, b)$, for $1 \leq k \leq m$, $1 \leq j \leq n$, there are two cases. If $\text{indegree}(R_{m+n+1}) = 2$ then using eq. (4.5),

$$\text{Cost } t(R_{m+n+1}, S_{k,j,1}, a) = \text{Cost } t(R_{m+n+1}, S_{k,j,1}, b) = \text{Cost } t(R_k, S_{k,\emptyset,\emptyset}, a) + \bar{a}(S_{k,\emptyset,\emptyset}) + \text{Cost } t(R_{m+j}, S_{\emptyset,k,\emptyset}, b) + \bar{b}(S_{\emptyset,k,\emptyset}).$$

If $\text{indegree}(R_{m+n+1}) = 1$ then using eq. (4.6)

$$\text{Cost } t(R_{m+n+1}, S_{k,j,1}, b) = \text{Cost } t(R_k, S_{k,j,1}, a) + \bar{a}(S_{k,j,1}).$$

Note that, the immediate predecessor of R_{m+n+1} in $t(R_{m+n+1}, S_{k,j,1}, b)$ must be R_k and the set $S_{k,j,1}$ remains unchanged in the substrategy with result node R_k (by property 6). The cost of the substrategy $t(R_k, S_{k,j,1}, a)$ is expressed in terms of the costs of $t(R_{k-1}, S_{k-1,j,1}, a)$ and $t(R_{m+n+1}, S_{k-1,j,1}, a)$ as given above. The cost of $t(R_{m+n+1}, S_{k,j,1}, a)$ can be computed similarly.

The costs of the optimal strategies over the sets $S_{k,\emptyset,1}$, $S_{\emptyset,j,1}$, $S_{k,\emptyset,\emptyset}$ and $S_{\emptyset,j,\emptyset}$ are computed in linear time. For example, consider $t(R_{m+n+1}, S_{k,\emptyset,1}, b)$. R_{m+n+1} is a single attribute relation in $S_{k,\emptyset,1}$ but it is a multiattribute relation in the next higher level strategy since the attribute b is common to R_{m+n+1} and its immediate successor. If $|R_{m+n+1}[a]| \geq |R_k[a]|$ then by property 7,

$$\text{Cost } t(R_{m+n+1}, S_{k,\emptyset,1}, b) = \text{Cost } (R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_k \rightarrow R_{m+n+1}).$$

If $|R_{m+n+1}[a]| < |R_k[a]|$ then eq. (4.4) applies resulting

$$\text{Cost } t(R_{m+n+1}, S_{k,\emptyset,1}, b) = \text{MIN } \{\text{Cost } t(R_k, S_{k,\emptyset,\emptyset}, a) + \bar{a}(S_{k,\emptyset,\emptyset}),$$
$$\text{Cost } t(R_k, S_{k,\emptyset,1}, a) + \bar{a}(S_{k,\emptyset,1})\},$$

where

$$\text{Cost } t(R_{m+n+1}, S_{k,\emptyset,\emptyset}, a) = \text{Cost } (R_1 \rightarrow \ldots \rightarrow R_k),$$
$$\text{Cost } t(R_{m+n+1}, S_{k,\emptyset,1}, a) = \text{Cost } (R_1 \rightarrow \ldots \rightarrow R_{m+n+1} \rightarrow \ldots \rightarrow R_k)$$

and $R_{m+n+1}[a]$ appears in size order in $t(R_{m+n+1}, S_{k,\emptyset,1}, a)$.

The boundary conditions are

$$\text{Cost } t(R_1, S_{1,\emptyset,\emptyset}, a) = \text{Cost } t(R_{m+1}, S_{\emptyset,m+1,\emptyset}, b) =$$
$$\text{Cost } t(R_{m+n+1}, S_{\emptyset,\emptyset,1}, a) = \text{Cost } t(R_{m+n+1}, S_{\emptyset,\emptyset,1}, b) = \emptyset.$$

Thus, with the exception of certain end conditions like Cost $t(R_{m+n+1}, S_{k,j,1}, a)$ with indegree$(R_{m+n+1})=2$, it takes only a constant number of operations to compute the optimal strategies at the stage (k,j) if the optimal strategies at the stages $(k-1,j)$ and $(k,j-1)$ are known. Since there are $m.n$ stages, the total number of operations required to compute the optimal strategy is $O(m.n)$ while there are at least $O(\binom{m+n}{m})$ potentially optimum strategies (see example 4.10) for the class of queries considered above. However, for general tree queries finding the optimal strategy may take exponential time.

4.4.3 Optimum Strategy Fully Reducing one of the Relations

Consider a query Q such that each relation referenced by Q has a distinct output attribute. Suppose the result node does not contain any of the relations referenced by Q. The optimal way to obtain the answer of the query Q at the result node is to use an optimum sequence of semi-joins fully reducing all the relations referenced by Q and then send the projections over the output attributes to the result node. Let R_i be the only relation fully reduced with respect to Q by a strategy $T(R_i,S)$. By Prop. 4.1, there exist a tree query graph QG of an equivalent query such that

$T(R_i,S)$ traverses QG from the leaves to the root R_i. Let S have N, $N \geq 2$, relations. A sequence of N-1 semi-joins, z^*, which traverses QG from the root R_i back to the leaves, if preceeded by $T(R_i,S)$, fully reduces the remaining relations [3]. z^* has the property that for each semi-join $R_j \dashrightarrow R_k$ in z^*, R_j is fully reduced with respect to Q before the semi-join is executed. This corresponds to the minimum amount of data transfer required for the semi-join $R_j \dashrightarrow R_k$.

Let $A(S) = \{a_1,\ldots,a_n\}$ be the set of all attributes referenced by the qualification of Q. Clearly, each attribute a_j in A(S) corresponds to a connected component in JG_Q. Let m_j be the number of vertices in the connected component of JG_Q corresponding to the attribute a_j, $1 \leq j \leq n$. Since QG is a tree query graph of an equivalent query, there is a spanning forest JG of JG_Q^T representing that equivalent query (by Lemma 4.2). There are m_j-1 edges in the connected component of JG corresponding to the attribute a_j. By Lemma 4.1(a), QG contains exactly m_j-1 edges between relations having attribute a_j in common. Let (R_1, R_k) be one of those m_j-1 edges in QG. z^* either contains $R_1 \dashrightarrow R_k$ or $R_k \dashrightarrow R_j$. In either case the cost of the semi-join is $\bar{a}_j(S)$ since the semi-join $R_i \dashrightarrow R_k$ (resp. $R_k \dashrightarrow R_i$) is executed after R_i (resp R_k) is fully reduced with respect to Q and Q is a conjunction of equi-join clauses. Since the same arguments are true for every attribute in A(S) and z^* contains exactly one semi-join for each edge in QG,

$$\text{Cost}(z^*) = \sum_{j=1}^{n}(m_j-1).\bar{a}_j(S).$$

Clearly, the cost of z^* is the same irrespective of which relation in S is the first fully reduced relation. Consider any other sequence z which is preceeded by $T(R_i,S)$ and fully reduces the remaining N-1 relations such that R_i is transmitted to the other relations via some semi-joins. The semi-joins in z form a transmission path from R_i to every other relation R_j, $j\neq i$, $1\leq i,j\leq N$. Then z traverses at least a spanning tree of QG^*. By Lemma 4.3(b), any spanning tree of QG^* is a tree query graph QG' of an equivalent query. Similar to QG, for each attribute a_j in A(S), QG' contains m_j-1 edges which are between relations having attribute a_j in common. Consider one of such edges (R_i, R_k) in QG' and let $R_i \dashrightarrow R_k$ be the corresponding semi-join in z. The $\text{Cost}(R_i \dashrightarrow R_k) \geq \bar{a}_j(S)$ since R_i may not be fully reduced before $R_i[a]$ is transmitted to R_k. Since the same arguments are applicable to every semi-join in z corresponding to the edges of QG', Cost(z) can not be lower than $\text{Cost}(z^*)$. That is, z^* has the minimum cost to fully reduce the remaining N-1 relations after R_i is fully reduced by $T(R_i,S)$ for any R_i in S.

Let t(S) be the optimum strategy fully reducing one of the relations in S. It is straightforward to show that for any two single attribute relations R_i and R_j in S having attribute a in common, Cost $T(R_i,S)$ < Cost $T(R_j,S)$ if

$|R_i[a]| > |R_j[a]|$. Thus, in order to find t(S) it is sufficient to consider the set of all optimal strategies $T(R_k,S)$, where R_k is either a multiattribute relation in S or $R_k = \max(a_j,S)$ for $a_j \in A(S)$, and then choose the one with minimum cost among those strategies. Consequently, an optimal strategy fully reducing all the relations in S consists of t(S) followed by z^*.

4.4.4 An Example

In order to illustrate the procedure to find the optimum strategy fully reducing a relation at the result node, we consider a distributed database with four relations,

R_1: EMPLOYEE (E#, Ename, Sex)

R_2: STUDENT-COURSE(E#, C#)

R_3: COURSE(C#, Cname, Level)

R_4: TEACHER-COURSE(E#, C#, Room).

This is the same example used in [13]. Suppose each relation resides at a separate site and the site containing R_1 is the result node. Consider the following query Q,

> "find the employee numbers of the employees who are students in a course and teaching an advanced level course".

The qualification of Q is

$(R_1.E\# = R_2.E\#).\underline{and}.(R_2.E\# = R_4.E\#).\underline{and}.$

$(R_4.C\# = R_3.C\#).\underline{and}.(R_3.Level = \text{"Advanced"}).$

The target list of the query is $R_1.E\#$.

After local processing, the selection clause (R_3.Level = "Advanced") is eliminated and each relation can be projected over its attributes referenced by Q. That is, the set S of relations after local processing is

$$S = \{R_1(E\#),\ R_2(E\#),\ R_3(C\#),\ R_4(E\#,C\#)\}.$$

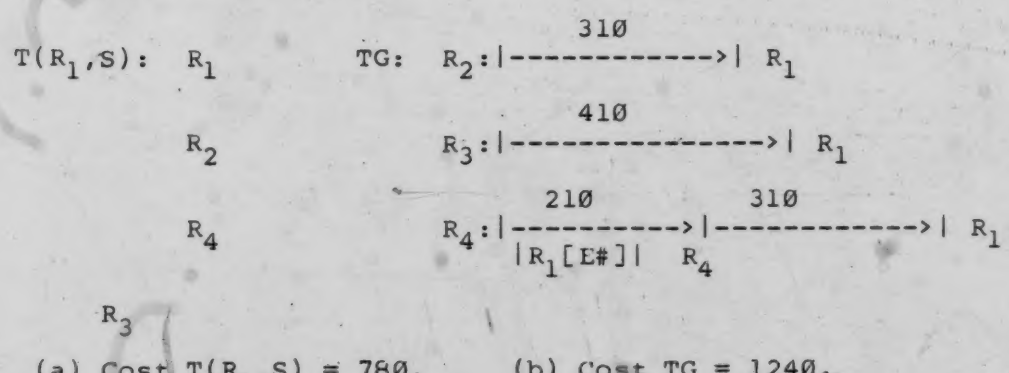
Let s_i denote the size of relation R_i, $1 \leq i \leq 4$. Suppose the sizes and the selectivities of the relations are as given in Figure 4.15, where the selectivity $p_{iE\#}$, i=1,2,4, is the probability that $R_i[E\#]$ contains a given value of the domain corresponding to E#. The selectivity $p_{iC\#}$, i=3,4, is defined similarly.

Let the cost of transmitting X amount of data from one site to another be C(X)=10+X. For estimating the size of a relation with more than one attribute, e.g. $R_4(E\#,C\#)$, the assumption in [13] will be used to facilitate a meaningful comparison between the algorithm presented in Section 4.4.2 and the one presented in [13]. Thus, after the semi-join $R_3 \dashrightarrow R_4$, where R_3 and R_4 have attribute C# in common, the estimated size of $R_4[C\#]$ is $|R_4[C\#]| * p_{3C\#} = 300 * 2/3$ while the size of $R_4[E\#]$ is unchanged.

The optimal strategy $T(R_1,S)$ fully reducing R_1 with respect to Q, computed using equations (4.3)-(4.6), is shown in Figure 4.16, where

	s_i	$\|R_i[E\#]\|$	$p_{iE\#}$	$\|R_i[C\#]\|$	$p_{iC\#}$
R_1	200	200	1/2	---	---
R_2	300	300	3/4	---	---
R_3	400	---	---	400	2/3
R_4	600	200	1/2	300	1/2

Figure 4.15. The Sizes and the Selectivities of the Relations.



(a) Cost $T(R_1,S) = 780$. (b) Cost TG = 1240.

Figure 4.16. The Optimal Strategy $T(R_1,S)$ and the Strategy TG.

$Cost(R_3 \rightarrow R_4) = 10 + |R_3[C\#]| = 10 + 400 = 410.$

$Cost(R_4 \rightarrow R_2) = 10 + |R_4[E\#]| = 10 + 200 = 210.$

$Cost(R_2 \rightarrow R_1) = 10 + |R_2[E\#]| * p_{4C\#} = 10 + 300 * 1/2 = 160.$

Thus,

$Cost\ T(R_1,S) = 410 + 210 + 160 = 780.$

Since E# is the output attribute and R_1 resides at the result node $T(R_1,S)$ is the optimum strategy to answer the query.

The strategy TG found by using the algorithm given in [13] (namely, Algorithm G) to answer Q is shown in Figure 4.16. The cost of transferring R_2 to R_1 is 310, the cost of transferring R_3 to R_1 is 410 and the cost of transferring R_4 to R_1 is 210+10+600/2 = 520. Hence, the total cost of TG is 410+310+520 = 1240, which is significantly higher than the cost of the optimal strategy $T(R_1,S)$ for this example.

CHAPTER 5

CONCLUSIONS

In this thesis, two related distributed relational database query processing problems have been studied, and solutions are proposed. The first problem is to determine whether or not a distributed query belongs to a special class of queries, called tree queries. Tree queries can always be answered by using only semi-joins which, in turn, are usually computed with much less data transmission than joins. Thus, it is important to be able to determine the tree-query membership of a distributed query.

The second problem studied in this thesis is distributed query optimization for tree queries. This is an important special case of the general query optimization problem for which there is no known optimum solution.

For the tree query membership problem, distributed database queries that are conjunctions of join and selection clauses with relational operators $\{>, \leq, =, \neq, <, \geq\}$ have been considered. A canonical representation of a distribued

query, called reduced join graph, has been introduced. The query represented by a reduced join graph does not contain any redundant join clauses, and is equivalent to the original query. It has been shown that the set of equivalent queries represented by reduced join graphs sufficiently characterize the set of all equivalent queries for determining tree query membership of a given query. A conceptually simple and efficient algorithm has been presented which determines whether or not such a query is a tree query. For tree queries, the presented algorithm produces an equivalent query whose query graph is a tree, and outputs the equivalent query together with its tree query graph so that a sequence of semi-joins to answer the original query can immediately be obtained. Thus, the presented algorithm can effectively be utilized to improve the performance of distributed query processing mechanisms. An implementation of the algorithm has been outlined and its time and space requirements have been established. The extension of this work for queries involving disjunctions is straightforward since such queries can be transformed into disjunctive normal form and each conjunction can be treated as a separate query. The tree query membership algorithm could be further extended for more general queries (e.g. queries with embedded quantifiers).

In general, there are numerous semi-join sequences, called strategies, that can be used to answer a tree query, and these strategies may differ significantly in the amount

of required data transmission. In order to find the optimum strategy, first a set of properties has been given to eliminate the strategies that can never be the optimum, then a dynamic programming approach has been presented which finds the optimum strategy among the potentially optimum ones.

Distributed query optimization for tree queries presented in this thesis uses only one size estimation assumption which is the uniformity and independence of values drawn from the same domain. This assumption leads to a reasonably good estimation for the size of the relation's column on which the semi-join is performed. One of the important problems to be solved in distributed query processing is to find a good method to estimate the sizes of other columns of a relation after one of its columns are reduced. The proposed estimation methods in the literature involve inaccuracies, which may significantly reduce the effectiveness of the query optimization. A novelty of the optimization presented in Chapter 4 is that it is independent of the choice of the method used to estimate the sizes of other columns of a relation when one of its columns is reduced.

The distributed query optimization presented in this thesis is for a class of tree queries, i.e., queries with equi-join clauses where the number of common joining attributes between any two relations in the query is at most

one. This work may be extended for more general tree queries. Hopefully, further generalization leads to optimal query processing strategies for distributed queries in general.

BIBLIOGRAPHY

[1] Aho, A.V., Garey, M.R. and Ullman, J.D., "The Transitive Reduction of a Directed Graph", SIAM Journal of Computing, Vol. 1, No. 2, 1972, 131-137.

[2] Babb, E.,"Implementing a Relational Database by Means of Specialized Hardware", ACM Trans. Database Syst. Vol.4, No.1 (March 1979), 1-29.

[3] Bernstein, P.A. and Chiu, D-M. W., "Using Semi-joins to Solve Relational Queries", Technical Report no. CCA-79-01, Computer Corporation of America, 1979.

[4] Bernstein, P.A. and Goodman, N., "Full Reducers for Relational Queries Using Multi-attribute Semi-joins", Technical Report no. TR-10-79, Aiken Computation Lab., Harvard University, Cambridge, Mass., 1979.

[5] Chiu, D.M. and Ho, Y.C., "A Methodology for Interpreting Tree-Queries into Optimal Semi-join Expressions", Proc. ACM SIGMOD Int. Conf. on Man. of Data, 1980, 169-178.

[6] Codd, E.F., "A Relational Model for Large Shared Databanks", CACM 13, 6, 1970, 377-387.

[7] Codd, E.F., "Relational Completeness of Database Sublanguages", Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, N.J., Prentice Hall, 1971.

[8] Codd, E.F., "Extending Database Relational Model to Capture More Meaning", ACM Trans. Database Syst., Vol.4, No.4 (Dec., 1979), 397-434.

[9] Date, C.J., An Introduction to Database Systems, Addison-Wesley, Reading, MA, 1977.

[10] Epstein, R., Stonebraker, M. and Wong, E., "Distributed Query Processing in a Relational Database System", ACM SIGMOD, Int. Conf. on Management of Data, 1977, 169-180.

[11] Goodman, N. et al., "Query Processing in SDD-1: A System for Distributed Databases", Tech. Rep. CCA-79-06, Computer Corporation of America, 1979.

[12] Harary, F., Graph Theory, Addison-Wesley, Reading, Mass., 1969.

[13] Hevner, A. and Yao, S.B., "Query Processing in Distributed Database Systems", IEEE Trans. on

Software Engineering, Vol. SE-5, No. 3, 1979, 177-187.

[14] Hsiao, D. and Menon, J., "Postprocessing Functions of A Database Machine", Tech. Rep., Ohio State University, Columbus, Ohio, July 1979.

[15] Munro, I., "Efficient Determination of the Transitive Closure of a Directed Graph", Information Processing Letters, Vol.1, (1971) 56-58.

[16] Ozkarahan, E.A., Schuster, S.A. and Sevcik, K.C., "Performance Evaluation of a Relational Associative Processor", ACM Trans. Database Syst. Vol.2, No.2 (June 1977), 175-195.

[17] Özsoyoğlu, Z.M. and Yu, C.T., "On identifying a class of Distributed Queries that can be Processed Efficiently", (submitted for publication), 1980.

[18] Pecherer, R.M., "Efficient Evaluation of Expressions in a Relational Algebra", Proc. ACM Pacific Conf., 1975, 44-49.

[19] Rothnie, J.B. and Goodman, N., "A Survey of Research and Development in Distributed Database Management", Proc. International Conference on Very Large Databases, Tokyo, 1977, 48-62.

[20] Rothnie, J.B., et al., "Introduction to a System for Distributed Databases(SDD-1)", ACM Trans. Database Syst., Vol. 5, No. 1 (March 1980), 1-17.

[21] Smith, J.M. and Chang, P.Y-T., "Optimizing the Performance of a Relational Algebra Interface", CACM, Vol. 18, No. 10 (Oct. 1975), 568-579.

[22] Stonebraker, M. and Neuhold, E., "A Distributed Database Version of INGRES", Berkeley Workshop on Dist. Data Management and Computer Networks, Berkeley, 1977.

[23] Su, Y.W. and Emam, A., "CASDAL: CASSM's Data Language", ACM Trans. Database Syst. Vol.3, No.1 (March 1978), 57-91.

[24] Ullman, J.D., Principles of Database Systems, Computer Science Press Inc., Reading, Maryland, 1980.

[25] Warshall, S.,"A Theorem on Boolean Matrices", JACM Vol.9 (1962), 11-12.

[26] Wong, E., "Retrieving Dispersed Data from SDD-1: A System fo Distributed Databases", Berkeley Workshop

on Dist. Data Management and Computer Networks, Berkeley, 1977.

[27] Wong, E. and Youssefi, K., "Decomposition-A Strategy for Query Processing", ACM Trans. Database Syst., Vol. 1, No. 3 (Sept. 1976), 223-241.

[28] Yao, S.B., "Optimization of Query Evaluation Algorithms", ACM Trans. Database Syst., Vol. 4, No. 2 (June 1979), 133-135.

[29] Yu, C.T. and Özsoyoğlu, Z.M., "An Algorithm for Tree-Query Membership of a Distributed Query", Proc. of IEEE Third Int. Conf. on Comp. Software and Applications, Nov. 1979, 306-312.

[30] Yu, C.T. and Özsoyoğlu, Z.M., "On Determining Tree-Query Membership of a Distributed Query", ACM Trans. Database Syst., (to appear) 1980.

[31] Yu, C.T., Lam, K., and Özsoyoğlu Z.M., "Distributed Query Optimization for Tree-Queries", (submitted for publication) 1980.

APPENDIX-A:

This appendix formalizes the transitive reduction of an acyclic join graph that is utilized in Section 3.2 for obtaining the reduced join graph of a query. The construction of the transitive reduction of an acyclic join graph is also given and its time complexity is established.

A.1. Transitive Reduction of an Acyclic Join Graph

Let G be the condensed acyclic graph for the join graph JG_Q of a query Q. Observe that, if JG_Q is acyclic then G denotes JG_Q itself. Informally, the transitive reduction G^t of G is a directed graph that does not have any redundant arcs while having the same transitive closure as G, i.e. $(G^t)^T=G^T$[1]. The transitive reduction in our context is a direct extension of the one in [1], for acyclic join graphs that contain three types of arcs. The union and the intersection operations on directed graphs are extended for join graphs as given below.

A.1.1. Basic Definitions

Let T={0,1,2,3} be the set of types of arcs in G where 0 denotes the type of a nonexisting arc (i.e. if there is no arc from u to v, it is considered as a type-0 arc). We define a binary addition operator, $\oplus$, and a binary multiplication operator, *, over the set T with the addition and multiplication tables as follows:

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	1	2	2
2	2	2	2	2
3	3	2	2	3

*	0	1	2	3
0	0	0	0	0
1	0	1	2	0
2	0	2	2	0
3	0	0	0	0

Clearly, $\oplus$ is associative and commutative, 0 is the identity of $\oplus$, i.e. $i \oplus 0 = 0 \oplus i = i$, for all $i \in T$, and T is closed under $\oplus$. Likewise * is associative and commutative, T is closed under *, and * distributes over $\oplus$. Observe that, for any two arcs $(u,v)_i$ and $(v,u)_j$ in G, $i \oplus j$ is the type of the arc from u to v that dominates $(u,v)_i$ and $(u,v)_j$. By convention, the sum over an empty set of arcs is 0.

As discussed previously (Section 2.2), each arc $(u,v)_i$, $1 \leq i \leq 3$, in G, is a type-i path of length 1 from u to v. A sequence of arcs (v_0,v_1), (v_1,v_2),..., (v_{k-1},v_k), $k \geq 2$, is a path from v_0 to v_k of length k if none of the arcs in the path is type-3. The type of such a path is 1 if every arc (v_{i-1},v_i), $1 \leq i \leq k$, is a type-1 arc. Otherwise it is a type-2 path. Observe that the product of the types of arcs in a sequence of arcs $p = (v_0,v_1)$, (v_1,v_2),...,(v_{k-1},v_k) gives the type of the path from v_0 to v_k if the arcs in p form a path; otherwise the product is 0 indicating that the arcs in p does not form a path from v_0 to v_k. Since G is acyclic and contains no self loops, the length of any path in G is between 1 and n-1, where n is the number of vertices in G, and the type of every path is well-defined.

We now give the extended union and intersection operators, denoted $\underline{\cap}$ and $\underline{\cup}$ respectively, on join graphs (see Figure A.1). Let G_1 and G_2 be any two finite acyclic join graphs[+] having the same vertex sets. Then

$$G_1 \underline{\cap} G_2 = \{(u,v)_i \mid (u,v)_j \in G_1,\ (u,v)_k \in G_2 \text{ and } i=j\oplus k,\ 1\leq j,k\leq 3\}$$

$$G_1 \underline{\cup} G_2 = \{(u,v)_i \mid (u,v)_i \in G_1 \underline{\cap} G_2\} \cup \{(u,v)_i \mid (u,v)_i \in G_1 \text{ and } (u,v)_j \notin G_2 \text{ for any } 1\leq j\leq 3\} \cup \{(u,v)_i \mid (u,v)_i \in G_2 \text{ and } (u,v)_j \notin G_1 \text{ for any } 1\leq j\leq 3\}$$

A.1.2. Transitive Reduction

For a given finite acyclic join graph G, let S(G) be the set of all join graphs having the same transitive closure as G, i.e. $S(G) = \{G_1 \mid G_1^T = G^T\}$. The transitive reduction G^t of G is defined as

$$G^t = \underline{\bigcap}_{G_i \in S(G)} G_i$$

Proposition A.1: For any finite acyclic join graph G, the transitive reduction G^t is unique and satisfies

(i) $(G^t)^T = G^T$ and

(ii) if $H^T = G^T$ then for any arc $(u,v)_i \in G^t$, there is an arc $(u,v)_j \in H$, $1\leq i,j\leq 3$.

[+] Throughout the Appendix, a graph G is represented as a set containing all the arcs of G. The symbols $\cup$, $\cap$ and $-$ are used to denote the usual set union, set intersection and set subtraction operators, respectively.

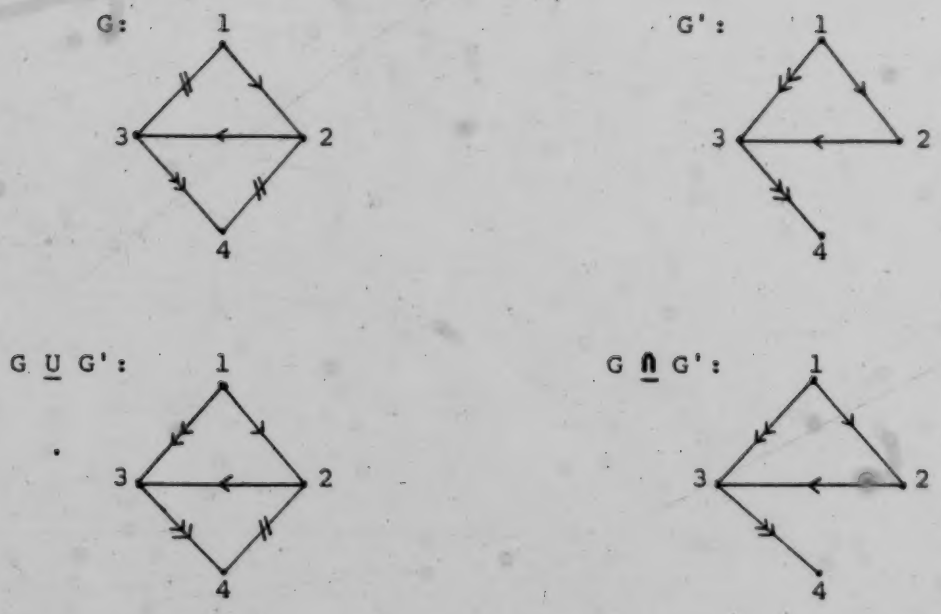


Figure A.1: The Operators $\underline{U}$ and $\underline{\cap}$.

Proof: Follows from the following three lemmas. (Note that Lemma A.1 and Lemmas A.2 and A.3 are actually straightforward extensions of Lemma 1 and Lemma 2 in [1].) In Lemmas A.1 and A.2 below, G_1 and G_2 denote any two finite acyclic join graphs (over the same vertex set) satisfying $G_1^T = G_2^T$.

Lemma A.1: If there is an arc $a = (u,v)_i$, $1 \le i \le 3$, in G_1 such that there is no arc from u to v in G_2, then

$$(G_1 - \{a\})^T = G_1^T = G_2^T = \{G_2 \cup \{a\}\}^T$$

where u and v are any two vertices in G_1.

Proof: Since $a \in G_1$, there is an arc $b = (u,v)_k$ in G_1^T, $1 \le k \le 3$. Since $G_1^T = G_2^T$, $b \in G_2^T$. Then the type of b is either 1 or 2 since G_2 does not contain any arc from u to v and no type-3 arc can be implied by a path from u to v of length ≥ 2 in G_2.

Case 1: $b = (u,v)_1$. Then $a = (u,v)_1$ and there is no (implied or not) type-2 path from u to v in G_1 or G_2. Since $b \in G_2^T$ there is at least one type-1 path in G_2 from u to v passing through some other vertex w. Thus $(G_2 \cup \{a\})^T = G_2^T$. Moreover G_1 also contains a type-1 path from u to w and a type-1 path from w to v. Since G_1 is acyclic, neither the path from u to w nor the path w to v can contain $a = (u,v)_1$. Thus $G_1 - \{a\}$ also contains a type-1 path from u to v, and $(G_1 - \{a\})^T = G_1^T = G_2^T$.

Case 2: $b = (u,v)_2$. The type of a may be 1, 2 or 3.

Since $b \in G_2^T$, there is at least one type-2 (or implied type-2) path in G_2 from u to v passing some other vertex w. Thus $(G_2 \cup \{a\})^T = G_2^T$. With no loss of generality let the path from u to w be type-2 (or implied type-2). Thus $(u,w)_2 \in G_2^T$ and $(w,v)_i \in G_2^T$, $1 \leq i \leq 2$. Then $(u,w)_2 \in G_1^T$ and $(w,v)_i \in G_1^T$ since $G_1^T = G_2^T$. G_1 contains a type-2 (or implied type-2) path from u to w and a type-i path from w to v. Since G_1 is acyclic, none of the paths from u to w and from w to v in G_1 contains a. Thus $G_1 - \{a\}$ also contains a type-2 (or implied type-2) path from u to v, and $(G_1 - \{a\})^T = G_1^T = G_2^T$. #

Lemma A.2: If $(u,v)_i \in G_1$ and $(u,v)_j \in G_2$ where $1 \leq i,j \leq 3$ and $G_1^T = G_2^T$ then

$$((G_1 - \{(u,v)_i\}) \cup \{(u,v)_k\})^T = G_1^T = G_2^T$$

where $k = i \oplus j$.

Proof: For k=i the result follows immediately. Since i=j implies k=i, assume $i \neq j$ and $k \neq i$. Then k=2 since $i \oplus j = 2$ for $i \neq j$, $1 \leq i,j \leq 3$. Moreover $(u,v)_i \in G_1$, $(u,v)_j \in G_2$, $i \neq j$, and $G_1^T = G_2^T$ implies $(u,v)_2 \in G_1^T$. G_1 contains a type-2 (or an implied type-2) path from u to v independent of the type of $(u,v)_i \in G_1$. Thus the arc $(u,v)_i$ in G_1 can not contribute to a type-1 or a type-3 arc in G_1^T. Therefore $((G_1 - \{(u,v)_i\}) \cup \{(u,v)_k\})^T = G_1^T = G_2^T$. #

Lemma A.3: Let G be a finite acyclic join graph. Then the set $S(G) = \{G_1 \mid G_1^T = G_2^T\}$ is closed under $\underline{\cap}$ and $\underline{\cup}$.

Proof: Let G_1 and G_2 be any two numbers of S(G), i.e.

$G_1^T=G_2^T=G^T$. Let $A=\{a_1,\ldots,a_x\}$, $B=\{b_1,\ldots,b_y\}$ and $C=\{c_1,\ldots,c_z\}$ be the set of arcs where

$A = \{(u,v)_i \mid (u,v)_i \in G_1$ and $(u,v)_j \notin G_2$ for any j, $j\le i, j\le 3\}$

$B = \{(u,v)_i \mid (u,v)_i \in G_2$ and $(u,v)_j \notin G_2$ for any j, $j\le i, j\le 3\}$

$C = \{(u,v)_k \mid (u,v)_i \in G_1$ and $(u,v)_j \in G_2$, $k=i \oplus j$, $j\le i, j\le 3\}$

By definition $G_1 \underline{\cup} G_2 = A \cup B \cup C$ and $G_1 = C$. There is one-to-one correspondence between the arcs in G_1-A and the arcs in C such that $(u,v)_i \in G_1-A$ iff $(u,v)_k \in C$ where $k=i \oplus j$ and $(u,v)_j \in G_2-B$.

Let $G_1-A=\{a_1,\ldots,a_z\}$ corresponds to $c_i \in C$, $1\le i\le z$. From Lemma A.2

$$((G_1-\{a_1\}) \cup \{c_1\})^T = G_1^T$$

$$((((G_1-\{a_1\}) \cup \{c_1\}) - \{a_2\}) \cup \{c_2\})^T = G_1^T$$

Hence $((G_1 - \{a_1,a_2\}) \cup \{c_1,c_2\})^T = G_1^T$.

By successive applications of Lemma A.2 , we have

$$((G_1-\{a_1,\ldots,a_z\}) \cup \{c_1,\ldots,c_z\})^T$$
$$= (A \cup C)^T = G_1^T = G_2^T = G^T.$$

Similarly $(B \cup C)^T = G_2^T = G^T$. Thus $(A \cup C)^T$ $= (B \cup C)^T = G^T$.

For any arc $a=(u,v)_i$ in A, $1\le i\le 3$, there is no arc from u to v in $B \cup C$. Then, by successive applications of Lemma A.1 for every arc in A

$$((A \cup C) - \{a_1\})^T = (B \cup C \cup \{a_1\})^T = G^T$$

$$(((A \cup C) - \{a_1\} - \{a_2\})^T = (B \cup C \cup \{a_1,a_2\})^T = G^T$$

$$\vdots$$

$$((A \cup C) - A)^T = C^T = (B \cup C \cup A)^T = G^T.$$

Since $G_1 \underline{\cup} G_2 = A \cup B \cup C$ and $G_1 \underline{\cap} G_2 = C$,

$(G_1 \underline{\cup} G_2)^T = (G_1 \underline{\cap} G_2)^T = G_2^T$, i.e.

$G_1 \underline{\cup} G_2 \in S(G)$ and $G_1 \underline{\cap} G_2 \in S(G)$. *

Since $S(G)$ is a finite set, proposition A.1 follows from Lemma A.3. (Note that proposition A.1 is a straightforward extension of Theorem 1 in [1] for the transitive reduction in our context). Thus from Proposition A.1, the transitive reduction of an acyclic join graph G can be obtained by successively examining each arc in the transitive closure of G and eliminating those arcs which are implied by transitivity.

A.2. Computing the transitive reduction

The transitive reduction of an acyclic join graph containing more than one connected component simply consists of the transitive reductions of its connected components. Thus, it is sufficient to discuss computing the transitive reduction of an acyclic join graph that consists of one connected component. Let G denote such a join graph with n vertices and A be the adjacency matrix of G, where

$$A(i,j) = \begin{cases} 1 & \text{if } (i,j)_1 \in G \\ 2 & \text{if } (i,j)_2 \in G \\ 3 & \text{if } (i,j)_3 \in G \text{ or } (j,i)_3 \in G \\ 0 & \text{otherwise} \end{cases}$$

for $i \neq j$, $1 \leq i,j \leq n$, and $A(i,j)=0$ for $1 \leq i \leq n$. Note that a type-3 arc $(i,j)_3$ in G is represented by two entries, namely, $A(i,j)=A(j,i)=3$ in A. Since G is acyclic, the vertices in G may be renamed so that every entry below the diagonal in the corresponding adjacency matrix A is either 0 or 3 (this conversion can easily be performed in time bounded by n^2 [15]). Consider a nonzero entry in A that is below the diagonal, i.e. $A(i,j)=3$, where $i>j$ and $1 \leq i,j \leq n$. Such an entry in A does not contribute to any path in G since the arc $(i,j)_3$ is also represented by $A(j,i)=3$, and there is no path from i to j in G for any $i>j$, $1 \leq i,j \leq n$. Consequently the entries in A that are above the diagonal are sufficient to represent all the arcs in G (i.e. with no loss of generality all the entries that are below the diagonal after the conversion can be considered as zero).

In order to compute the transitive reduction of an acyclic join graph G, first the vertices in G are renamed so that the corresponding adjacency matrix is upper triangular, as discussed above. The next step is to compute the transitive reduction G'^t of the resulting graph G'. Finally the transitive reduction G^t of G can be obtained from G'^t by restoring the original names of the vertices. Since the first and the last steps are straightforward and take at most $O(n^2)$ time, we now discuss obtaining the transitive reduction of a join graph whose adjacency matrix is upper triangular.

Let G be a finite acyclic join graph whose adjacency matrix A is upper triangular. The algorithm presented below computes the transitive reduction G^t of G.

Algorithm T:

1) Find the transitive closure G^T of G and let A^T denote the adjacency matrix of G^T.
2) Compute $A_1=AA^T$ and let G_1 be the graph whose adjacency matrix is A_1.
3) Then G^t is $G^T - G_1$.

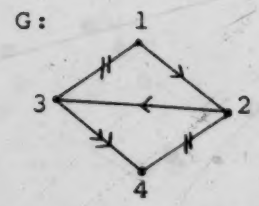
An example illustrating Algorithm T is given in Figure A.2.

We use the following notation. AB denotes the multiplication of two (nxn) adjacency matrices, A and B, using the multiplication operator * and the addition operator $\oplus$. That is, C=AB means $C(i,j)=(A(i,1)*B(1,j))\oplus\ldots\oplus(A(i,n)*B(n,j))$ for $1\leq i,j\leq n$. Similarly, $A\oplus B$ denotes the addition of A and B such that $C=A\oplus B$ means $C(i,j)=A(i,j)\oplus B(i,j)$ for $1\leq i,j\leq n$. Now it remains to establish the time complexity and the correctness of Algorithm T.

Since G is acyclic, the adjacency matrix A^T of the transitive closure of G is given by

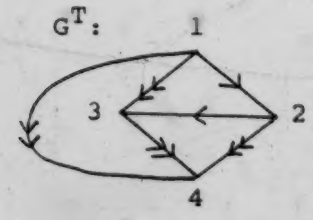
$$A^T=A \oplus A^2 \oplus\ldots\oplus A^{n-1}$$

where $A^k=A^{k-1}A$ for $1<k<n$. It is clear that A^T can be computed in $O(n^3)$ time using an algorithm similar to Warshall's Algorithm [25]. Since the complexity of step 2 is



$$A = \begin{bmatrix} 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

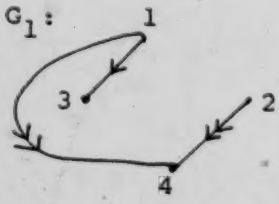
(a) Acyclic Join Graph G and its Upper Triangular Adjacency Matrix A.



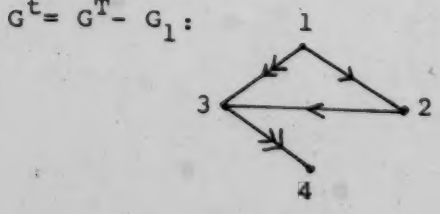
$$A^T = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) Transitive Closure G^T and its Adjaceny Matrix A^T (step 1).

$$A_1 = A\,A^T = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



(c) G_1 and its Adjacency matrix A_1 (step 2)



(d) Transitive Reduction G^t (step 3)

Figure A.2: An Example for Algorithm T.

proportional to n^3, and that of step 3 is bounded by n^2, the overall complexity of Algorithm T is $O(n^3)$.

The correctness of Algorithm T follows from Lemma A.4. which shows that G^T-G_1 is the transitive reduction G^t of G.

Lemma A.4: Let G be an acyclic join graph whose adjacency matrix A is upper triangular. Then $G^t=G^T-G_1$, where G_1 is the graph whose adjacency matrix $A_1=AA^T$ and "-" denotes set subtraction.

Proof: By definition $A^T=A\oplus A^2\oplus\ldots\oplus A^{n-1}$. From the distributive property of * over $\oplus$, $A_1=A^2\oplus\ldots\oplus A^{n-1}$. Clearly, an arc $a=(u,v)_i$ is in G_1 iff there is a path of length ≥ 2 from u to v in G implying the arc $(u,v)_i$ without using the arc from u to v in G, $1\leq i\leq 3$. Thus an arc $a=(u,v)_i$ is in G^T but not in G_1 iff none of the paths of length ≥ 2 from u to v in G implies the arc a without using the arc from u to v in G. Since this is exactly the condition under which an arc $a=(u,v)_i\in G^t$, we have $G^T-G_1=G^t$. #

Thus, the transitive reduction G^t of an acyclic join graph having one connected component can be computed in $O(n^3)$ time, where n is the number of vertices in G. Consequently, the transitive reduction of an acyclic join graph having m connected components each with n_i vertices, $1\leq i\leq m$, takes at most $O(\sum_{i=1}^{m} n_i^3)$ time.